Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Vesa Paakkanen

User Experience of Game Development Environments:

Can making games be as fun as playing them?

Master's Thesis
Espoo, September 10th, 2014

Supervisor: Professor Marko Nieminen
Instructor: Jyrki Korpi-Anttila Lic.Sc. (Tech.)

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Vesa Paakkanen

Title:
User Experience of Game Development Environments: Can making games be as fun as playing them?

Date:	September 10th, 2014	**Pages:**	vii + 71
Major:	Software Business and Engineering	**Code:**	T-76
Supervisor:	Professor Marko Nieminen		
Instructor:	Jyrki Korpi-Anttila Lic.Sc. (Tech.)		

Game industry has grown to be the biggest entertainment industry during the past few years. This has lead to exponential growth of size and complexity of game projects. Good development tools are required in order to work with such complexity. However, the available tools do not meet the expectations of the game teams, even though better tools could save both a lot of time and money.

A study about the current state of user experience of game development environments was conducted for this thesis in a game development company. Feedback and experiences were gathered from the members of the game teams using a questionnaire, interviews and shadowing. The collected data was analysed qualitatively classifying the data and the results from the methods. Based on these results improvements were made to the tools used in the company.

Based on the results we concluded that the user experience of the game development environments can be significantly improved from their current state. The improvements should focus on allowing the teams to enable faster development iteration speed as well empowering content creators to work without the help of programmers.

Keywords: usability, user experience, game development, games, development enviroment, IDE

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Vesa Paakkanen		
Työn nimi:	Pelikehitystyökalujen käyttäjäkokemus: Voiko pelien tekeminen olla yhtä hauskaa kuin niiden pelaaminen		
Päiväys:	10. syyskuuta 2014	**Sivumäärä:**	vii + 71
Pääaine:	Ohjelmistotuotanto	**Koodi:**	T-76
Valvoja:	Professori Marko Nieminen		
Ohjaaja:	Tekniikan lisensiaatti Jyrki Korpi-Anttila		

Peliteollisuudesta on kasvanut suurin viihdeteollisuuden ala viime vuosien aikana. Siitä johtuen peliprojektit ovat kasvaneet eksponentiaalisesti sekä koossa että kompleksisuudessa. Hyviä työkaluja tarvitaan entistä enemmän sen kokoisen kompleksisuuden kanssa työskentelyyn. Nykyiset työkalut eivät kuitenkaan vastaa pelitiimien odotuksia, vaikka paremmat työkalut voisivat säästää sekä aikaa että rahaa.

Tätä diplomityötä varten tehtiin tutkimus pelikehitysympäristöjen käyttäjäkokemuksesta eräässä peliyrityksessä. Pelitiimien jäseniltä kerättiin palautetta ja kokemuksia käyttäen kyselylomaketta, haastatteluja ja varjostusta. Kerätty data analysoitiin laadullisesti luokittelemalla materiaalit ja kerätyt tulokset. Näiden tulosten perusteella yrityksen työkaluihin tehtiin parannuksia.

Tulokset osoittivat että pelinkehitysympäristöjen käyttäjäkokemusta voidaan parantaa huomattavasti nykyisestä tilasta. Parannusta kaivataan ainakin kehityksen iteraationopeuteen sekä mahdollistamaan sisällöntuottajien työskentelyn ilman ohjelmoijien apua.

Asiasanat:	käytettävyys, käyttäjäkokemus, pelikehitys, pelit, kehitysympäristö, IDE
Kieli:	Englanti

Acknowledgements

I would like to thank the Case Company for giving me an opportunity and plenty of time to work on this thesis project. I'd like to give special thanks to both of my instructors Lauri Alakuijala and Jyrki Korpi-Anttila, as well as the whole technology team in the Case Company. Thank you Antti A, Antti R, Christina, Henri, Johannes, Jon, Jouni, Juan, Mika, Osmo, Remi and Samuel. I would also like to thank Professor Marko Nieminen for supervising this thesis and for providing his advice and expertise that helped me greatly during the writing and research process.

Finally, I want to thank my friends and family for providing support and bearing with me during this time. Thank you Tuuli, Juho and Tomi.

Helsinki, September 10th, 2014

Vesa Paakkanen

Abbreviations and Acronyms

API	Application Programming Interface
DLL	Dynamic-link library
FPS	First-Person Shooter
GUI	Graphical User Interface
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LOC	Lines Of Code
QA	Quality Assurance
SDK	Software Development Kit
SE	Software Engineering
SUS	System Usability Scale
SVN	Subversion (Version Control System)
UDK	Unreal Development Kit
UI	User Interface
UX	User Experience
VCS	Version Control System
XML	eXtensible Markup Language

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Problem . . . 2
- 1.3 Research objectives and approach . . . 2
- 1.4 Scope . . . 3

2 Game Development **4**
- 2.1 Evolution of game development . . . 4
  - 2.1.1 Code-Driven development . . . 8
  - 2.1.2 Data-driven development . . . 9
- 2.2 Game Development Environment . . . 10
  - 2.2.1 Components of a Game Development SDK . . . 10
  - 2.2.2 Common tools . . . 11
  - 2.2.3 Available game development SDKs . . . 11
- 2.3 Game Development Phases . . . 13

3 Research Environment: The Case Company **16**
- 3.1 The Company Profile . . . 16
- 3.2 Users: Game Team . . . 17
  - 3.2.1 Designers . . . 17
  - 3.2.2 Developers . . . 17
  - 3.2.3 Artists . . . 18
  - 3.2.4 Producers . . . 18
- 3.3 Supporting teams . . . 18
  - 3.3.1 Technology team . . . 18
  - 3.3.2 Operations . . . 19
  - 3.3.3 Quality Assurance . . . 19
  - 3.3.4 Business Intelligence . . . 19
  - 3.3.5 Digital Services . . . 19
- 3.4 Communication between teams . . . 20
- 3.5 Game Development Environments in the Case Company . . . 20

3.5.1 Unity . . . 20
3.5.2 Fusion . . . 22
3.6 Game Development Culture . . . 22

4 Research process and methods **25**
4.1 Previous work . . . 25
4.1.1 The definition of User Experience . . . 25
4.1.2 Academic papers . . . 26
4.1.3 Industry articles and presentations . . . 26
4.2 Workshop in Technology Team . . . 27
4.3 Questionnaire . . . 27
4.4 Background interviews . . . 28
4.5 Observation of usage of Game Development environments . . . 29

5 Results **30**
5.1 UX according to Technology Team . . . 30
5.2 Results of SUS-questionnaire . . . 32
5.2.1 Fusion . . . 33
5.2.2 Unity . . . 38
5.3 User Experience for different roles in the team . . . 42
5.4 Observed usage of the tools . . . 45
5.4.1 Team using new version of Fusion . . . 46
5.4.2 Team using old version of Fusion . . . 47
5.4.3 Team using Unity . . . 48
5.5 Fusion design and implementation . . . 49

6 Evaluation **55**
6.1 Research Objectives . . . 55
6.2 Changes in technology development in the Case Company . . . 56

7 Conclusion and Discussion **58**
7.1 Validity and reliability . . . 58
7.2 The current state in the Case Company . . . 60
7.3 Improving the current state . . . 61
7.4 Future work . . . 63

A Interview Questions **68**

B Games Unit Questionnare **70**

Chapter 1

Introduction

This thesis researches how user experience of the current game development tools is perceived and how it could be improved. It also presents examples how tools evolve based on the feedback from the users and how the results affect the user experience.

1.1 Background

Game industry has grown to be the biggest entertainment industry during the past few years. The biggest entertainment launch to date was *Grand Theft Auto 5*, making more than one billion dollars during the first three days, which is almost the same amount that the entire music industry makes in a month[1]. The fastest growing part of the games industry has been mobile games, with games like *Clash of Clans, Candy Crush Saga* and *Angry Birds* being played by millions of people daily[2]. For example, Angry Birds has been downloaded more than billion times to date[3].

Games are being made using different *game development environments*. The development environment always contains a game engine and some tools to work with the engine. The environment may be a single integrated development environment (IDE) like Unity, or it may consist of several different tools, it may even be as simple as a text editor used to write scripts for the game. The game development environment usually abstracts the low-level implementations such as rendering, input and networking to higher level pro-

[1]http://www.metalinjection.net/videogames/grand-theft-auto-v-outsells-entire-music-industry, retrieved on 23.05.2014

[2]http://thinkgaming.com/app-sales-data/, retrieved on 23.05.2014

[3]http://www.rovio.com/en/news/blog/162/1-billion-angry-birds-downloads/2012, retrieved on 23.05.2014

gramming interfaces and tools . This allows the game teams to concentrate on writing the game logic and creating content for the game.

1.2 Problem

Due to the growth of the industry the complexity and size of game projects has grown exponentially during the past decades (Blow 2004). Also the number of platforms has grown significantly, especially on the mobile market. There are several hundreds of mobile devices with different specifications and it is extremely difficult to test that the game works and looks the same on all the platforms (Sididris 2014). This means that the cost of game development has been risen and developers need to find ways to reduce the costs (Folmer 2007).

Good development tools are required in order to work with such complexity, but most of the available tools do not meet the expectations of the game teams (Ahern 2012; Blow 2004; Lightbown 2013). However, it is difficult to design tools for the game teams as game development involves many people with different skills. Since artists, designers, developers and testers all have varying needs and expectations about the tools they use in their daily work, providing tools with a consistent user experience for all the members of the team is challenging. The deadlines for game projects are also notoriously demanding, which adds to the disregard of developing good tools when the production of the game title in time is the priority (Ahern 2012).

1.3 Research objectives and approach

The main objective of this thesis is to help better understand the needs of the game teams in the Case Company and to improve the quality of their daily work. Improving the tools should allow the teams to produce more polished games in shorter time frames as the tools should reduce tedious and repeating work. This thesis should identify the bottlenecks in game development tools and present potential solutions.

As secondary objectives this thesis presents some information on how some tools or platforms are adopted based on the user experience (i.e. what kind of correlation is there between the ease of development for a platform and the amount of software it has). In addition, it should present results if the user experience of the development tool can affect the resulting product (in this case the games).

The study in this thesis is done in two parts. The first part includes

a literature research into user experience, game development and game development tools. The second part of the study is done in a mobile game company, referred as the Case Company, by observing the development of the games and by interviewing the employees of the company about their usage of the game development tools.

1.4 Scope

The thesis concentrates on researching how two different game development environments are used by several game teams in a single mobile game company, referred to as the Case Company. The focus of the study is to find out what the user experience of these environments is like and how it could be improved. The emphasis will be on mobile game development, but the differences between mobile, console and PC game development will be investigated.

Chapter 2

Game Development

This chapter gives an overview about game development, how it has changed during the past decades and how the tools and processes have evolved. A few chosen game engines are also described in more detail.

2.1 Evolution of game development

Game development has existed as an industry since early 1970s with arcade machines, early game consoles and personal computers. Since then there have been several major changes in the industry, the latest of which is the rise of the mobile game market. The next change is expected to occur with the release of virtual reality accessories like Oculus Rift[1] and Sony's Morpheus[2] as the virtual reality offers the possibility to create new kind of gaming experiences.

Most video games can be considered *soft real-time interactive agent-based computer simulations* (Gregory 2009). In other words, most games run a real-time simulation of some kind of world and in that world several different entities interact dynamically.

Standard game project size and complexity have grown exponentially (Blow 2004). Early games like *Pong*[3] or *Asteroids*[4] could fit their world in a single screen and only contained simple sounds, 2D graphics and simulation in addition to a simple game logic. In comparison, a current Massive Multiplayer Game (MMG), e.g. *Elder Scrolls Online*[5] contains a vast 3D world

[1] http://www.oculusvr.com/, retrieved on 25.6.2014

[2] http://www.techradar.com/reviews/gaming/project-morpheus-1235379/review, retrieved on 25.6.2014

[3] http://www.mobygames.com/game/`pong_____`, retrieved on 25.6.2014

[4] http://www.mobygames.com/game/`asteroids_`, retrieved on 25.6.2014

[5] http://elderscrollsonline.com/, retrieved on 25.6.2014

with thousands of simultaneous players. The difference in the amount and quality of the *assets* (models, textures, sounds etc.) can be seen from the size of the games. The early games could fit in a few kilobytes whereas the current games may take more than 50 gigabytes (i.e. the capacity of a single Blu-ray disc).

We can see the increase and size in complexity by inspecting the source code of popular first-person shooter (FPS) games by id Software and Battlefield 3[6] by DICE we can see the growth in the amount of code during the past two decades. Figure 2.1 shows the number of lines of code (LOC) in game and engine projects and the clean installation size of the games on PC. The LOC is calculated from the source code of the games available on GitHub[7] by using **cloc**[8] with the exception of Battlefield 3. id Software has not released the source code of their latest game, Rage[9], and thus Battlefield 3 was selected as an example of a recently released game as an approximation of the size of its codebase was readily available[10].

Game	Lines of Code	Installation Size
Wolfenstein 3D (1992)	27 300	2 MB
Doom (1993)	39 420	13 MB
Quake (1996)	78 961[11]	80 MB
Quake 2 (1997)	163 928	370 MB
Quake 3 (1999)	367.815	490 MB
Doom 3 (2004)	593 832	1.5 GB
Battlefield 3 (2011)	1 500 000	11.6 GB
Doom 3 BFG Edition (2012)	318 517	11 GB
Engine: UDK 4 (2014)	1 738 328	N/A
Engine: Fusion (2014)	207 240	N/A

Figure 2.1: FPS Game sizes.

Figure 2.2 shows us that the LOC of the games have been steadily growing. The amount of code has almost doubled in every subsequent game by id Software and when comparing the earliest game, Wolfenstein 3D (Figure 2.3), to Battlefield 3 (Figure 2.4) the size of the codebase is more than fifty

[6]http://www.battlefield.com/battlefield3, retrieved on 25.6.2014

[7]https://github.com/id-Software/, retrieved on 5.6.2014

[8]http://cloc.sourceforge.net/

[9]http://bethsoft.com/en-gb/games/RAGE, retrieved on 25.6.2014

[10]http://dice.se/publications/parallel-futures-of-a-game-engine-v2-0/, retrieved on 5.6.2014

[11]Excluding QuakeWorld source code as it was not part of the original release

times larger. On the other hand the size of the codebase of Doom 3 BFG Edition is about half of the size of Doom 3 even though BFG Edition is an improved version of the game and also targeting console platforms in addition to PC. The analysis of the differences between the two codebases is out of the scope of this thesis but the differences are probably a contribution of several things, such as improved skills of the programmers, advances in the programming language (C++ in this case) and more time to polish the codebase. Most importantly, the difference shows that the amount of code is not always a reliable metric of the complexity of a game.

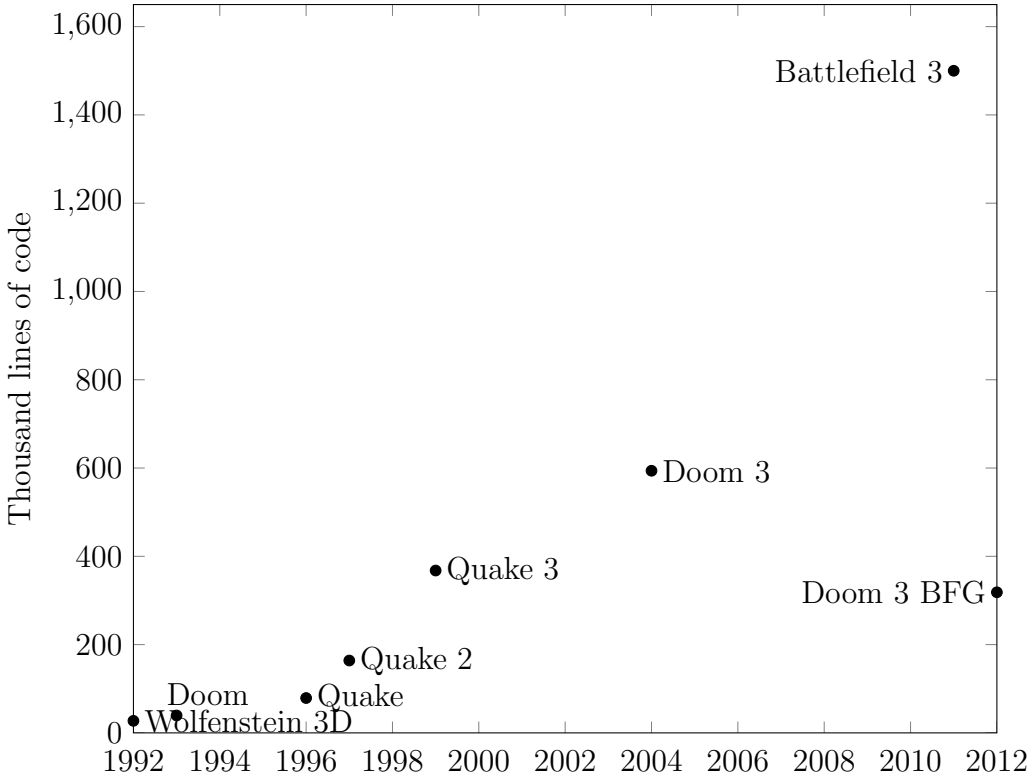

Figure 2.2: Growth of amount of code in FPS games.

The installation size of the games in Figure 2.1 shows how the number and fidelity of the assets has increased during the years. As an example the resolution of a wall texture in Wolfenstein 3D was 64x64 pixels while textures used in more modern games can be up to 4096x4096, in other words the textures may be up to 4096 times larger than twenty years ago. In addition the games contain, for instance, 3D models, speech, music and animations. All these types of assets are usually created by professionals specialized on

Figure 2.3: Wolfenstein 3D by id Software.

that type. The amount and size of raw assets (i.e. assets produced by content creation tools such as Photoshop or Maya) is huge in modern games. For example, during production Battlefield 3 had 500 GB[12] and Rage 1 TB[13] of raw uncompressed assets. This increase in the amount and fidelity obviously means that more skill and time are required to produce high quality assets.

Doom 3 BFG Edition, which has improved assets compared to the original game, shows how the size of the assets hs increased. It is almost ten fold compared to the original release. It should be noted, however, that the BFG edition also contains extra content (more levels as well as Doom 1 and Doom 2), but those additions should count only for a small fragment of the size of the game. Doom 3 BFG Edition demonstrates how the advancements in game development (programming languages, techniques, skills etc.) have enabled games to be built with less code, but also it shows how much larger the assets of the games have grown.

This growth in both content and logic has caused game development to shift from *code-driven development* towards *data-driven development*.

[12]http://dice.se/wp-content/uploads/Scaling-the-Pipeline.pdf, retrieved on 6.6.2014

[13]http://www.shacknews.com/article/53976/rage-will-look-worse-on, retrieved on 6.6.2014

Figure 2.4: Battlefield 3 by DICE.

2.1.1 Code-Driven development

In code-driven games the interactions and behaviors are created with code logic (Vachon 2013). Most of the very early games were written by a single programmer who did everything in code using some compiled language: the gameplay, the graphics and the sounds. The early games were written using Assembly but nowadays more high-level languages are used like C++, Objective-C or C# even though Assembly or C may still be used for optimization.

The heavy use of code means that the bulk of the work is done by programmers and most of the changes done by other members of the team have to go through them. Especially in big game projects this means that a lot of communication is required between the programmers and rest of the team. It also means the programmers may become a bottleneck in the production, and implementation of features may therefore take a long time. On the other hand in small projects the code-driven approach may be an advantage as the game is easy to debug and the programmers have virtually no limits on what can be added to the game.

The source code of Wolfenstein 3D[14] demonstrates the code-driven approach by having a list of sounds, maps and text of the game in the header-

[14]https://github.com/id-Software/wolf3d, retrieved on 6.6.2014

files instead of data-files. This means that every time the developers wanted to add a new map, sound or text they had to recompile the game. The actual graphics, sounds and maps are data (i.e. code just plays the sound asset rather than produces it) but references to the data are hard-coded.

2.1.2 Data-driven development

During mid- and late 1990s the complexity of the games started to grow significantly due to the 3D games becoming increasingly popular (Blow 2004). During this period the term *game engine* arose first in reference to the popular FPS games like Doom, Quake and Unreal. The software components of those games were separated and designed as reusable. This gave birth to "modding" (i.e. modifying the game to work differently, for example, by changing some game logic or changing the graphics or the sounds of the game) as well as engine licensing which in turn was the beginnings of the data-driven development.

According to Gregory, *data-driven architecture* differentiates a game engine from a game that is not an engine (Gregory 2009). In a data-driven architecture the behaviors and interactions of the actual game are created with data logic and the designers are making the most of the work (Vachon 2013). This essentially means that the engine itself does not contain any game specific rules, logic or special cases that prevent reusing the engine to create a different game. Some engines specialize in a specific genre of games while others allow creation of any game.

Obviously the more generic the engine is the less optimal it is to the specific requirements (Gregory 2009). Therefore modifying the engine the game can be improved. However, some of the commercial game engines do not allow changes to the source code or may not even provide it.

One of the first examples of a game engine before the term was used is *Script Creation Utility for Maniac Mansion* (SCUMM) which was created for Maniac Mansion in 1987 and used for several other adventure games by LucasArts for over ten years. It was essentially a script interpreter, built specifically to create 2D adventure games, that also initialized graphics and sounds, as well as read files from the disk. Because the scripts controlled everything in the game, it was easy to port to multiple platforms as the interpreter was the only part of the games requiring modifications. Even more importantly, scripting enabled fast prototyping and fast iteration by designers and artists with low development cost (Bevan 2013). Although SCUMM is over 25 years old, it was recently ported to iOS and Android. Furthermore, the fundamentals of it are the same as in modern game engines such as Unity, where the scripts control the game.

2.2 Game Development Environment

Currently, several engines, editors and tools specializing on different types of games are available. These cover different levels of users and teams with different sizes. In this thesis the term *game development environment* is used to refer to all of the software (i.e. SDKs and tools) which are used by the game teams to create games.

2.2.1 Components of a Game Development SDK

The game development SDK may consist of several different components. The core component of an SDK is the actual game engine but in addition to that the SDK may offer visual development tools. The main components of an SDK are described below.

Engine: A software framework for game development. The core functionality of a game engine includes, among others, graphics rendering, physics simulation, scripting, sound, animation, networking and cross-platform support. In other words the goal of the engine is to provide the common functionality for all the games, enabling game developers to concentrate on the main game logic. The concept of a data-driven engine is explained in detail in section 2.1.2.

Some engines are more limited than others and may concentrate only on a specific functionality such as rendering. On the other hand most of the engines can be extended to add more functionality.

Middleware: A piece of software provided by some third party that will be integrated into the game engine. It usually specializes in some aspect of the game engine, such as physics, rendering or UI. Examples of middleware are Havok Physics and Scaleform (used for building menus and UIs for games).

Editor: An application providing a graphical user interface (GUI) to work with the engine. The editor makes the creation of the content more accessible especially to artists and designers. It usually also provides other tools to help the development such as debugging, profiling and build tools. The editor may be implemented as a single IDE or as several separate tools. Editors are also often extendable similar to the enginesto enable game teams to modify the tools to suit better to the needs of a specific game.

2.2.2 Common tools

There are some tools that are commonly used in the game development. A few of the most common ones used in the Case Company are described below.

Microsoft Visual Studio: An IDE for Windows which supports several programming languages like C++, C#, Python and JavaScript. Visual Studio is commonly used for writing, compiling and debugging the native (C++) code of the games.

Apple Xcode: An IDE for OSX which is similar in features and supported languages to Visual Studio with addition of Objective-C which is used in applications on iOS and OSX-platforms. Xcode also provides an iOS simulator for testing iOS-applications without an actual device.

Adobe Photoshop: An image editing program used to draw sprites and textures to be used by the games.

Autodesk Maya: A 3D graphics software used to build different kinds of 3D assets, for example, character models and levels.

In addition of these tools the development teams virtually always use text editors, version control systems (VCS) and instant messaging tools of their choice. Subversion, Perforce and Avid by Alienware are popular version control systems due to their ability to manage binary assets.

2.2.3 Available game development SDKs

Currently there are several major SDKs available for game development. An SDK for game development usually offers a data-driven game engine and some IDE to work with the engine. Some SDKs are specialized for specific game type, while others are more generic. Here are some examples of currently popular SDKs:

Unity: Currently one of the most popular game engines, especially for indie and mobile game studios. In 2012 it was the most popular engine among mobile game developers, with 53.1% of developers reported using Unity in some way or the other. In comparison 39.8% reported using a custom engine (Deloura 2012). Unity is described in more detail in section 3.5.1.

Unreal Engine: Another popular game engine which is used in several AAA-games. The recently released version of the toolset, UDK4, aims

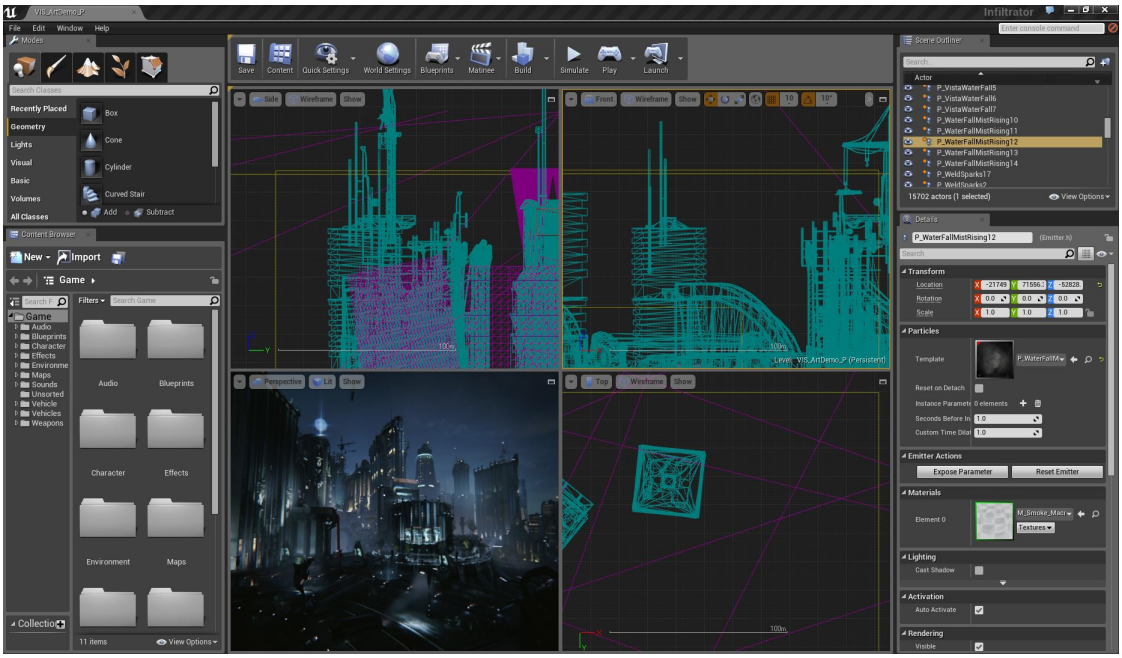

Figure 2.5: Unreal Editor 4 by Epic Games.

to make it more accessible to smaller studios and indie developers with a simple monthly subscription model and a percentage of the sales as royalties. This has enabled it to directly compete with Unity. Figure 2.5 shows the IDE of latest version UDK.

Cocos2D: A free cross-platform open source framework for building games and other graphical applications. The original Cocos2D was written in Python and supported only Windows, OS X and Linux. Currently the most popular versions of the framework are Cocos2D-Swift and Cocos2D-X. The former is written in Objective-C and is for iOS only while the latter is written in C++ and is cross-platform. There is also an editor available for Cocos2D-X, CocosStudio, that enables faster content creation.

RPG Maker: An editor allowing creation of role-playing games (RPGs) in a style of Japanese console RPGs released in the mid 1990s. The editor has been released for several different platforms including consoles like Super Nintendo and Sony Playstation. The specialization to a specific kind of games has enabled the editor to become accessible to less experienced users and it is currently marketed as "Simple enough for a

child"[15]. Figure 2.6 shows the latest version of the editor.

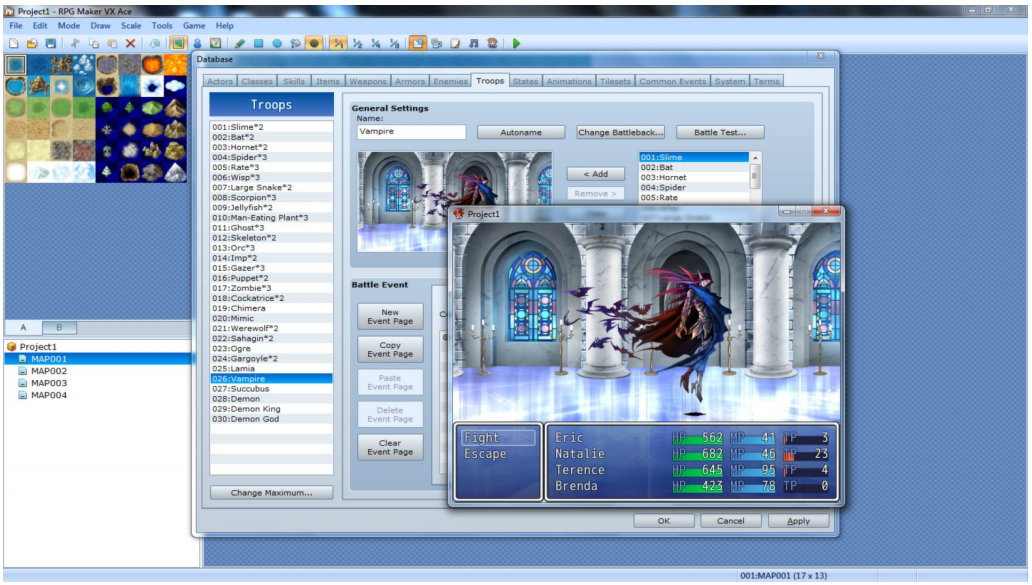

Figure 2.6: RPG Maker VX Ace by Enterbrain.

2.3 Game Development Phases

Game development process contains several phases. Different companies may obviously have different processes, and thus the phases may also vary. The phases of the development process below are listed according to Novak (Novak 2011).

Concept: The goal of the concept phase is to create a concept document describing what the the game is about and to communicate the vision of the game to others in written form. The team creating it is usually small and there isn't any actual development work done during this phase.

Prototype: The definition of prototype is defined by Novak as "a working piece of software that captures onscreen the essence of what makes your game special, what sets it apart from the rest, and what will make it

[15]http://www.rpgmakerweb.com/products/programs/rpg-maker-vx-ace, retrieved on 12.6.2014

successful" (Novak 2011). In other words, the goal during this phase is create a short version of the game demonstrating the core gameplay.

Sometimes it may be useful to create a paper-based prototype before or during the creation of the digital version (Novak 2011). This helps test the game mechanics with little effort, and is often faster than programming a digital version. The amount of work to be done to create a digital prototype can be greatly reduced if a game engine and/or tools are already done. However, the engine and the tools of the prototype may also be completely different than what the final game will be using.

The prototype may be the most important thing influencing whether or not the game gets to the production phase at all. The prototype is usually what the decision makers want to see and what convinces them to continue the development of the game past the prototype.

Pre-Production: During this phase the actual game development is done. Production may take from a few months to a few years depending on the game. This phase may be further divided to separate parts according to functionality or the content of the game, for example, production of single- and multiplayer or creation of different parts of the game world.

Localization: Some games need to be localized to different markets. This doesn't only mean translating the voice and text but it may also mean content modifications to match the regulations on different markets. In some cases even the gameplay may be changed in different countries. A game which is released as a premium (you pay up-front) title in Western markets may be a free-to-play title in China and the gameplay has to be modified accordingly.

Alpha: During this phase the game is playable from start to finish. All the assets may not be final, there are usually a large number of bugs left and some of the features may be missing. The goal is to polish and finalize the features in the game. The game may also be released to a small number of alpha testers who are not part of the actual development team.

Beta: The last phase before the release is beta phase. During this phase the game should contain the final assets and features. The focus is fixing bugs, performance testing/tweaking and making sure the game is stable. Some companies utilize crowd-sourcing for beta-testing. One

example of crowd-sourcing is the Early Access-program in Valve Software's Steam[16].

Gold: When a game "goes gold" it is considered to be finished and sent to manufacturing or to a digital marketplace. The manufacturing of physical media may take several weeks due to time required for creating the media and packaging it, whereas the release to a digital marketplace may take only a few days.

Post-Production: After the game is released it will be in post-production phase. During this phase *patches*, *updates*, *downloadable content (DLC)* and *expansions* are created. The patches usually fix software bugs, updates contain some additional content or features (as well as bug fixes). The DLC and expansions are purely additional content. The former are smaller packs, for example new characters or levels, and the latter are larger ones that may contain a whole new campaign.

[16]http://store.steampowered.com/genre/Early Access/, retrieved on 3.7.2014

Chapter 3

Research Environment: The Case Company

This chapter describes the game development process in the game company; what kind of departments the company has, what is the composition of the game teams, and how the teams communicate both internally and between different teams. The development culture will also be discussed.

3.1 The Company Profile

The company is one of the largest mobile game developers in Finland and has a few hundred employees working in the Games Unit. There are several game teams and supporting teams inside the Games Unit and the actual teams are relatively small compared to the size of the whole unit. There may be more than ten different game teams working on both new and existing games as well as creating prototypes at the same time.

Mobile game studios differ from AAA-game studios, in which there may be more than hundred people working on a single game and the company may only be working on one or two games at once. Also, a single AAA-game may take several years to make compared to a mobile game which on average takes about six months.

The Case Company is also unique compared to other game companies by having separate units working on other entertainment industries such as animation, consumer products and books. Some game launches are done together with these other units to bring game related products to the market. This cooperation between different industries also brings additional requirements for some games, for example, the game should be able to "scan" toys and provide the player with some additional value by having both the game

and the related toys.

3.2 Users: Game Team

An ordinary game team in the company contains about ten members. The team consists of a producer, (lead) designer, two to three programmers and two to three graphic artists. Some teams have dedicated level designer(s) and in some teams the lead designer builds the levels. There are only two sound artists in the company at the moment who are not part of any team but are involved in making the sounds and the music for all the games.

The main responsibility of the game team is the game itself, but for some games the team may need to integrate or build some services in the game. For example, some games require Facebook-integration. During the background interviews the programmers said that the integrations take up too much time from developing the actual game. In the future the responsibility of the integrations is moving to the supporting teams and are planned to be optional parts of the game engine.

3.2.1 Designers

The game team usually has one designer who is responsible for the overall game design. In some game projects there are additional level designers mainly responsible for creating levels for the game, but they may also participate in the overall game design.

The majority of the designers' work goes to planning and communicating the design to the game team. When working directly with the game they usually work with a tool that allows them to tweak the variables in the game in an easy way. Also when creating content, like levels, they use a visual tool.

Some designers have technical knowledge and are able to write some scripts, but it is unusual for a designer to have Visual Studio or other tool that allows compiling of the game so they are largely dependant on the developers for making any changes that require compiling.

3.2.2 Developers

There are usually two to three developers in a game team. They are responsible for writing the game logic code on top of a game engine. Depending on the game engine used, the game logic may be written as scripts, native code or both. In the game teams using Unity, the developers write "behaviour

components", which are essentially scripts, using C#. The game teams using the internal engine use Lua to write similar component scripts as in Unity, as well as use C++ to get access to more features of the engine besides the ones available to scripting.

The developers were the main driving force in the game teams in the company in the past. This is slowly changing, but still a lot of changes to the game have to go through the developers instead of other team members being able to make the changes themselves.

3.2.3 Artists

There are currently three different kinds of artists in the Case Company: graphics (i.e. sprites and textures), animation and sound artists. Some artists do both graphics and animations. The artists use Adobe Photoshop for creating the graphics, and Flash and Spine for creating animations. The assets are then exported into a format that the game engine understands.

3.2.4 Producers

Every game team has a producer. The responsibilities of a producer are similar to the role of a project manager in a traditional software engineering (SE) project, i.e. scheduling, resourcing and communicating the needs of the team to the supporting teams.

3.3 Supporting teams

The company also has support teams that are responsible for providing technology, tools and other services for all the game teams.

3.3.1 Technology team

The team responsible of developing and maintaining the internal game engine, editor, build tools and test automation tools. The team has about the same number of people as a single game team. Internally the people in the team are divided into four different teams: engine team, editor team, build system team and test automation team. Although these people have different responsibilities according to their team, they occasionally work on tasks outside their main responsibility (e.g. someone from the editor team may work on some engine features or vice versa).

All the members of the technology team are developers except for a project manager and a build engineer. Most of the people in the technology team are also more experienced than an average game developer in a game team.

3.3.2 Operations

The Operations team is responsible for porting and building the games to different platforms. The game team usually uses some platform, usually iOS, as their main platform that they develop on. The operations team then makes sure that the game compiles and works on other platforms and makes any necessary modifications. Operations team is also responsible for the localization of the games to different markets.

3.3.3 Quality Assurance

Quality Assurance (QA) team is responsible for testing the game. They create automated tests and also do manual testing. There is usually a single member of QA team assigned to each game team that has a game in production who is mainly responsible for testing the quality of the game. This includes both the technical quality (i.e. the game is not crashing) and the quality of the game experience (i.e. the game is fun to play).

3.3.4 Business Intelligence

The Business Intelligence (BI) provides analytics data for the game teams. The analytics provide information about the players, how active they are and what kind of devices they use. They also provide data about the game economy, for example, how many specific in-app purchases (IAP) are made and how much players spend in total. The data about economy is very important for the game teams when adjusting the economy of free-to-play titles to improve the monetization of the game titles.

3.3.5 Digital Services

Digital Services is a separate unit from the Games unit and it is dedicated for delivering an SDK for ads, social features, payment and analytics for the game teams as well as providing support during the integration of those features.

3.4 Communication between teams

The game teams are grouped into a few studios that each have a different theme. For example, one studio can concentrate on creating games for certain audience or demographic, another studio is working with certain brand while a third studio is producing more experimental games. The studio division does not prevent creation of games outside the theme of the studio, however the majority of the games by the teams in the studio respect the theme.

The teams inside a studio are mostly working in the same space, so the communication is simpler and more direct, because other team members are in close physical proximity. On the other hand, the communication between supporting teams happens mostly through Skype. Especially communication between game teams and the Digital Services-team (see 3.3.5) has proven to be challenging as they are working as separate unit.

Some studios have weekly meets for developers and/or artists to share knowledge, but in addition there are only a few events to share knowledge in the company, such as technology summit where the technology team informs the developers about the latest development in internal technology and provides a forum for the developers to share their experiences using the technology.

All teams have a dedicated wiki-page to inform people about the projects being done in the team, but in reality most of these pages are not actively maintained. However, test builds of the games are frequently released by the game teams and everyone in the other teams have read access to the source code repository of the games. This enables people to see the progress of the games and to test them out.

3.5 Game Development Environments in the Case Company

The company uses mainly two different toolsets: Unity and Fusion. These two SDKs with their surrounding environments in the Case Company are the focus of this study.

3.5.1 Unity

Unity is a cross-platform game engine with a built-in IDE by Unity Technologies. The development of Unity began in 2001 and it is still actively developed.

Figure 3.1: Unity Editor by Unity Technologies.

The engine of Unity is built using C++, but it is exposed to the developers through Mono (an open source implementation of .Net) with which the IDE of Unity is also built. This enables the developers to use the same language and the API to both build games and to extend the editor. The exposed interface allows game developers to write scripts as *behavior components*. These components are added to game objects in a scene (i.e. level) and they produce the interactions in the game. For example, game object called "character" may contain component that draws a character sprite and another component that listens to mouse input and starts moving the object towards a point where mouse was clicked. The scripts can be written either by using MonoDevelop which is part of the Unity installation, or with Visual Studio. The former is an open source IDE for writing C#-code and works on multiple platforms, while the latter is Windows-only (see Section 2.2.2).

Unity also uses middleware for several functionalities. Nvidias PhysX for 3D physics, Box2D for 2D physics, Mecanim for animations and Umbra for occlusion culling to name a few.

Asset Store for Unity was launched in 2010. It enables both developers and content creators to share and sell game assets and editor extensions to other Unity users. Currently thousands of assets are available for purchase in the asset store, varying from complete toolkits for specific game genres to

individual sprites or sounds.

The Case Company started to use Unity three years ago in a single game project to determine the feasibility of its use. Since the development of that particular game was a success, the usage of Unity has increased in the company and currently about half of the teams in the production phase use Unity.

3.5.2 Fusion

Fusion is an internal cross-platform game engine developed in the Case Company. The development began as a rewrite of the old game engine in 2012. The old engine was more of a framework providing cross-platform development and low-level functionality, such as rendering and input. The primary motivation of the rewrite was to modernize the engine by adding data-driven functionality using component-based system (entity-component system or ECS) similar to Unity to reduce the effort required by the game teams to create new games and prototypes.

The rewrite of the engine also allowed the Case Company to begin development of an editor on top of the engine. The goal of the editor is to provide a more accessible way to start building games with the engine and to allow less technical people to add and modify the content in the game. The functionality of the editor will be similar to Unity but as it is developed internally new features and fixes can be developed according to the needs of own game teams.

Currently both the engine and the editor are still early in development although development versions of both are used in a few game projects that are in early production phase.

3.6 Game Development Culture

Historically, the game development in the company has been more code-driven than data-driven. The old engine that was used internally only provided basic functionality such as rendering, sound, input and scripting support, so the game teams had to implement their own way of creating content for the games. In practice it meant that the game teams implemented game specific data structures and editors for their own game, and due to the time constraints the developers did not have enough time to provide artists and designers all the necessary tools to modify the game, so most of the changes had to go through the developers. For example, the artists drew the assets,

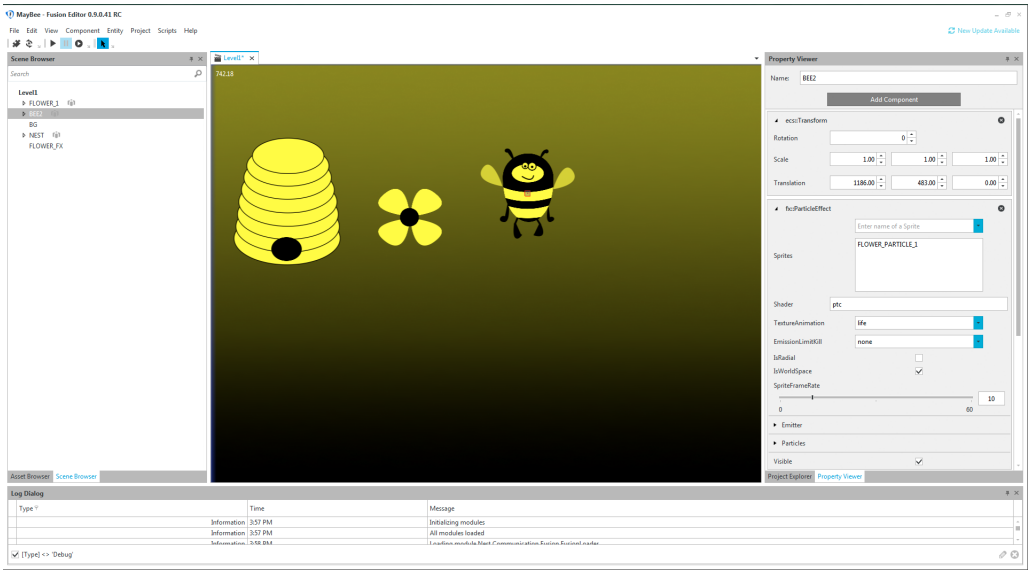

Figure 3.2: Development build of Fusion Editor.

but in most cases they had to have the developers add the assets to the actual game.

Currently all the games in production phase, with a single exception, are utilizing the data-driven functionality of either Unity or Fusion. This has enabled artists and designers to make changes to the games without having to involve developers at all. It has also enabled faster prototyping of game ideas in the company. There are still several games in post-production phase that use the old functionality as it would require too much effort to convert the games to use the new data-driven systems.

Figure 3.3 shows the difference in the amount of game logic compared to the other systems in two different games. The classic game does not use the data-driven functionality provided by the current version of the engine, while the new game uses the ECS-framework. It should be noted that the new game has not been released yet, but the developers do not expect the amount of the code the grow significantly as the game is only missing content (i.e. more levels) and is nearly finished in terms of functionality.

The Case Company has released several sequels to the classic game and the starting point of each sequel has been the codebase of the previous game. As the first game in the series (the classic game in figure 3.3) was developed without much consideration of possible further development, the codebase of the game was challenging to extend and maintain. Currently the game teams are still struggling with the codebase of the game series, as there are several

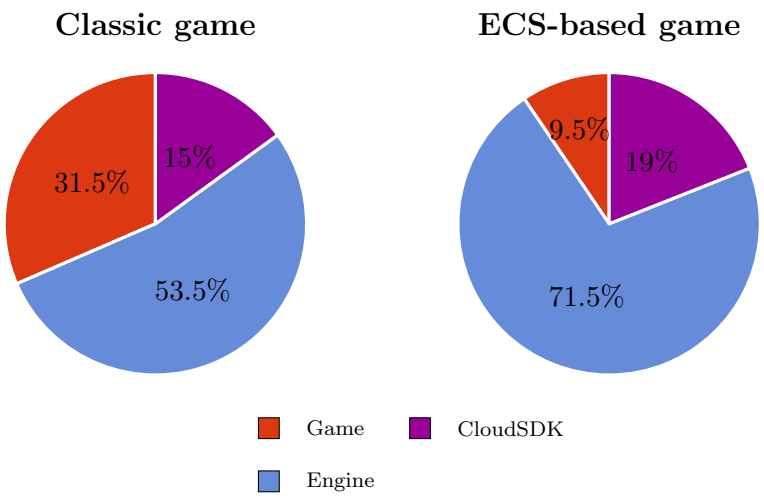

Figure 3.3: Amount of game logic in different game projects.

games using mostly the same codebase, each with some slight differences. Sometimes these differences cause problematic conflicts, such as a certain game requiring changes that would end up breaking the other games.

The most glaring problem with this method of copying the previous codebase to begin the development of a new sequel is that out of about 650 files in the codebase there are two files that comprise almost 15% of the LOC in the whole codebase. Modifying these files is extremely fragile and causes a lot of problems.

Chapter 4

Research process and methods

Since game development has only recently become a subject of academic research, only a small number of academic papers exist about the topic. Most of the work is done by gathering information from the employees of the company and doing empiric research.

4.1 Previous work

The first part of the study will be a study into previous research about the game development and the user experience. The goal is to combine the knowledge from the results of the academic research in User Experience with industry knowledge about the game development and the tools.

4.1.1 The definition of User Experience

In the past the focus of usability research has been about functionality and efficiency, but during the past decade the research has been shifted towards of the *experience* of using the product. The current ISO-standard of User Experience (UX) defines it as:

"Person's perceptions and responses that result from the use or anticipated use of a product, system or service."(ISO 2010)

There are also several other definitions for UX, which define it as dynamic, context-sensitive and subjective (Law et al. 2009). Due to the nature of the definition of UX being subjective it is difficult to evaluate objectively. For this reason, a definition of User Experience is provided that is used in the context of this thesis.

In this thesis, the term 'User Experience' is used to describe the experience of using game development tools. The scope of user experience is defined according to Hartson and Pyla (Hartson and Pyla 2012):

- effects experienced due to usability factors
- effects experienced due to usefulness factors
- effects experienced due to emotional impact factors

The main focus of this thesis is the usability and usefulness factors as the game development tools are primarily productivity tools built for development teams to create games. Usability and usefulness of the tools for collaboration in the team will be studied, i.e. how the tools affect the collaboration of people from different crafts and skills.

The emotional impact factors include aesthetics and expectations. Also, because the end result is an entertainment product, a game, fun will be investigated as an important part of the emotional impact factors, i.e. is the usage of the tools fun and can it be tested or designed using the tools.

4.1.2 Academic papers

User Experience is a widely researched subject and there is a lot of academic research material available (Law et al. 2009).

While there are several books about the game development in general, books concentrating on the tool development could not be found. However, some game development books, such as *Game Coding Complete*, contain some information about editor development (McShaffry 2013).

4.1.3 Industry articles and presentations

Even though there are only a small number of academic publications about the game development and even less about game development tools there are a lot of material available on game industry sites. The study in this thesis will concentrate on the material available on GDC Vault (*GDC Vault* 2014) and Gamasutra (*Gamasutra* 2014). The former contains Game Developers Conference-presentations about different topics of game development and the latter written articles. Both presentations and articles are performed and written by industry professionals.

The focus of the studied material will be on topics regarding game development tools, post-mortems and mobile game development.

4.2 Workshop in Technology Team

As the technology team (see section 3.3.1) is responsible for the development of the internal game development environment a simple survey will be conducted to determine which aspects the technology team considers to affect the UX of a game development environment and which of those aspects are the most important for a positive user experience.

The workshop will be conducted creating an affinity diagram about the user experience in the game development environment. The diagram will be constructed as described by Beyer and Holtzplatt (Beyer and Holtzblatt 1999). The team will start by compiling a list of items of the aspects affecting the UX of the game development environment and then writing those items on Post-it notes. The notes will initially be on the same level on a whiteboard and then each member can raise a single note one level up and lower another note one level down. This will continue for two rounds (i.e. each member has raised two and lowered two in total). As an additional rule the same person can't raise or lower the same note twice. This method was chosen as it is quick to perform and also you can ask why people raised or lowered a specific note and discussions can be initiated about the reasons. Additionally it shows the team if the members agree on what is important and what is not.

4.3 Questionnaire

To be able to get an overview of the usability of the used game development environments in the Case Company a questionnaire will be sent to the whole Games Unit.

The questionnaire utilizes System Usability Scale (SUS) (Brooke 1996). SUS questionnaire contains 10 questions presented as simple declarative statements, each with a five-point Likert scale from "strongly disagree" (value of 1) to "strongly agree (value of 5). It was selected as it is simple and easy to analyze and allows to compare the results between the development environments used in the Case Company. Due to the questionnaire being widely used tool to measure usability, it allows comparisons to other software systems. The questionnaire allows respondents to write a reasoning for the scoring to allow more accurate analysis of the score. The questions used in the questionnaire can be found in Appendix B.

The respondents will be divided into groups according to the role of the respondent to allow analyzing the usability from the perspective of different roles. Those groups will be further divided into 5 different subgroups where

a single respondent may belong to more than one group:

- Fusion User, with or without Unity experience
- Unity User, with or without Fusion experience
- Fusion User, without Unity experience
- Unity User, without Fusion experience
- User with experience from both Fusion and Unity

From each group the average, minimum, maximum and median scores will be calculated along with standard deviation. The grouping enables analyzing if the role of the respondent affects the SUS-scoring. For the first two groups the distribution of the scores will be calculated as well as the average score of each question in SUS-questionnaire.

4.4 Background interviews

To get more detailed information about the user experience of the game development environments the members of a single game team in the Case Company will be interviewed about the current state of the internal game development tools they are using currently. The goal of the interview is to find out what are the biggest bottlenecks in game development in the case company, how the current internal tools are perceived and what are the expectations of the new tools in development. The list of people to be interviewed is as follows:

- Two Fusion Programmers
- A Game Designer
- A Level Designer
- Two Artists (one graphics and one animation)
- A Producer

According to Hyysalo structured, themed or open interviews are possible (Hyysalo 2009). The first two types have a list of preselected questions that will be asked, but the themed one allows asking of additional questions. The open interviews are basically an open discussion about a selected topic. The selected method for the background interviews is the themed interview as it allows reaction to topics that weren't taken into account when creating the list of questions. The interviews will be conducted individually, so that the

answers will be more subjective. The questions used in the interview can be found in Appendix A.

The data gathered from the interviews will be analysed by Qualitative Data Analysis-method as described by Caudle (Caudle 2004).

4.5 Observation of usage of Game Development environments

Three game teams using different game development environments will be shadowed for a day. The goal of shadowing is to find out any problems that the game editors have and to see if people use them as expected. The focus of the shadowing is to observe which features of the editors are used and how. It is also used to determine if there are unnecessary features, features that the user cannot find, features that are difficult to use as well as find out which features work well.

The game environments used are the old version of Fusion without the component-based system, the new version of Fusion with the ECS-framework, and Unity. The Fusion-projects are in production phase while the Unity-project is in prototyping phase.

This method is considered an easy and cheap way to see how users think, especially if the users are willing to verbally explain what they think when using the environment (Ahern 2012). It also gives a chance to observe how the teams communicate internally. Some industry professionals consider it the only way to get good results (Acton 2013).

A few members of each game team will be shadowed, which according to Nielsen is enough to capture a large part of the usability problems (Nielsen 2012).

Chapter 5

Results

This chapter describes the results of the study using the methods described in the previous chapter.

5.1 UX according to Technology Team

In the beginning of the workshop the technology team discussed the aspects that contribute to the UX of a game development environment. The discussion was short as the members of the team were quick to come up with aspects and there were no disagreements. The resulted list of the aspects was as follows:

- Effectiveness (i.e. fast iteration time)
- Ease of use / Simplicity
- Stability
- Look and Feel
- Fun to use (inspires you to create and try out several things)
- Available knowledge (i.e. community, forums etc.)
- Documentation
- Frequent updates
- Ability to affect the development of the tool

With the exception of "effectiveness" being changed to more specific "fast iteration time", the same list of aspects was decided to be used in the interviews with the game teams (see section 4.4 and appendix A) to be able to

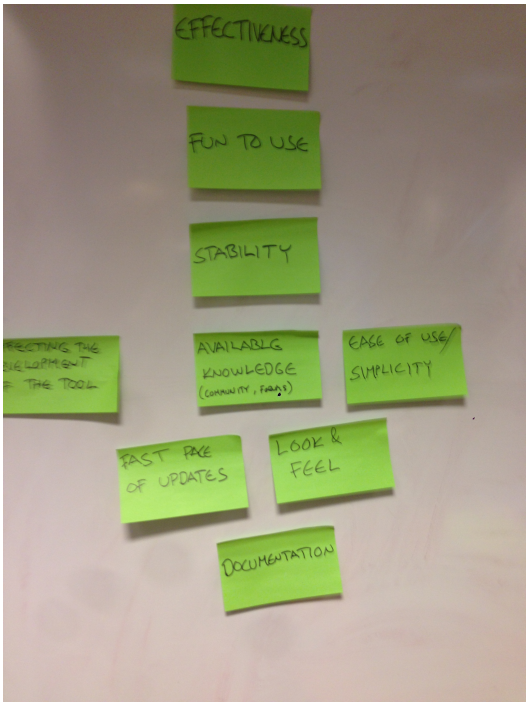

Figure 5.1: Result of the workshop in technology team.

determine if the opinions of technology team differ from the opinions of the game teams.

The most important aspect became clear during a few first turns of the first round as almost every member of the team considered effectiveness as an important aspect and raised it higher on the board. The overall top three aspects were established during the first round. During the second round the aspects below the top three were moved back and forth as members had some disagreements on the order of the rest of the aspects. The only aspect that stood out was documentation, which the members regarded as the least important. The overall result of the ordering can be seen in Figure 5.1.

The reasoning for considering effectiveness as the most important thing was due to the game development requiring a large amount of iterations to produce a high quality game. According to the team it is not only important that the game works technically (i.e. does not crash or have any bugs that cause undesired behavior) but it should be fun as well and the only way to create a fun game is by experimenting and polishing several ideas. The faster it is to try out different ideas the better the game will be.

The reasoning for "fun to use" being ranked as the second to most important item was that having an environment that is fun to use and which attracts people to try out different ideas and to play around supports the creative side of the game development.

Stability raised a discussion if it should have been ranked as the first due to having completely unstable environment causes the whole environment to be unusable. The team considered that it is acceptable for the environment to crash in rare cases, but it should be so stable enough to allow users perform any actions without fear of crashing the environment.

Documentation was ranked the least important as the team argued that if the environment is easy to use and there is a community around it there is no need for extensive documentation. The maintenance of the documentation requires a large amount of effort and if it is not up-to-date it will decrement the UX. However, the team pointed out that the importance of the documentation increases if the tools are provided by a third party instead of being developed internally, because the developers of third party tools are not as easily available to answer questions as internal developers are.

The team considered that it is important for game teams to be able to affect tool development, so that they are able to get tools that are made specifically for their needs, enhancing iteration time and simplicity.

The "look and feel"-aspect, aesthetics, also raised discussion if it should have been ranked as the least important aspect. It was argued that it does not provide any functional benefits, but on the other hand working daily with a visually displeasing environment can diminish the UX considerably.

The workshop lasted about an hour in total. Having a quick discussion about how well the current functionality reflects the desired level of the User Experience and how the current features can be improved according to the resulting list of the aspects was considered useful to have a quick discussion. The consensus in the team was that although the internal technology was heading in the right direction there is a large amount of things to be done before the UX of the internal technology is on the desired level.

5.2 Results of SUS-questionnaire

There were 67 respondents to the questionnaire, which is roughly 25% of the whole Games Unit and about 35% of the target audience of the questionnaire (i.e. after HR-people etc. are excluded from the headcount). The most active group of respondents was game developers, comprising about half of the respondents. Due to the low number of total artists responding to the questionnaire and having similar scores and reasoning they were aggregated into a

single group in the analysis of scoring. There were also only two respondents from Operations, and both of them had Fusion and Unity experience. They were therefore not divided into subgroups according to experience. The sizes of the different respondent groups can be seen in the Figure 5.2.

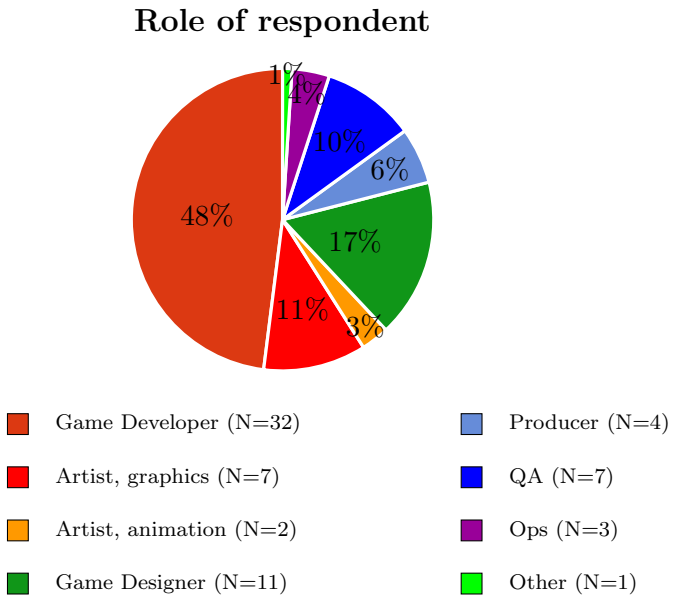

Figure 5.2: SUS-questionnaire respondents (N=67).

Both environments had low average scores. The average score of Fusion was 49 while score of Unity was 63,67. Fusion was expected to score lower than Unity as it is still in early development and Fusion Editor has not been released yet.

5.2.1 Fusion

A total of 35 respondents had used Fusion as their development environment. There were no significant differences in the average score of the different groups with the exception of Ops scoring Fusion considerably higher than other groups. The scores can be seen in Table 5.1.

Fusion received the most criticism from lacking a finished editor. The current editor was considered too unstable, missing a lot of critical features and having several usability problems. Another common criticism was the slowness of adding new features. The respondents thought that due to the technology team being small they sometimes have to wait for too long for

resources to be available to work on the issues the game team is having. Also, due to the large number of items to work on, the tech team is not able to polish the features, resulting in a feeling that something is always broken in some way. Fusion also received criticism that, unlike Unity, it does not have a unified environment. The respondents felt that it is difficult to work with several separate tools, especially with art pipeline. The list of items criticized can be seen in Table 5.2.

The developers gave the most positive feedback about Fusion, although they scored Fusion below the total average score. Fusion was said to have a good multiplatform-support and build tools. Especially, when compared to Unity Fusion was said to handle the more uncommon platforms such as BlackBerry better. Particularly, Operations team thought that Fusion far surpasses Unity in this aspect. In addition Fusion was praised for offering good performance and lots of scripting control by using C++ and Lua. Developers found that the ECS is a solid framework for building games, even if it still requires more work. The need for more work on the component system was the most common criticism from the developers.

The artists and designers found the existence of the editor positive and a big step up from previous methods, such as editing files with text editor. However, as the editor is still early in the development, they stated that in its current state it is not stable enough and can only be used for a few basic tasks. The list of of the best features in Fusion are in Table 5.3.

The distribution of the SUS-score for Fusion can be seen in Figure 5.3. The distribution suggests that the usability of the environment is considered poor company-wide. Furthermore, as there are no scores above 75 and only a few above 70, no one considers the level of usability good.

Surprisingly, Figure 5.4 shows that although the average SUS-score of Fusion was low, the average score of the first question, which states that a respondent would like to use Fusion frequently, was relatively high compared to to other questions. This may be explained by the fact that the game teams know that Fusion is still in development and will improve over time and by the desire to use internal technology, both of which are supported by the feedback received from the respondents.

The results in Figure 5.5 reflect the unfinished nature of Fusion. The respondents think that Fusion is not currently suitable for prototyping. The feedback strongly suggests this is due to the unfinished editor and ECS-framework. On the other hand respondents think that Fusion is somewhat suitable for production and it is supported by the fact that the older version has been successfully used in several released games.

Group	N	Mean	Min	Max	Std. Dev	Median
All	35	49	17,5	75	15,72	50
Developers	22	47,05	17,5	75	15,29	46,25
Designers	2	40	30	50	14,14	40
Artists	7	50,35	35	75	13,95	47,5
Operations	2	71,25	67,5	75	15,91	71,25
Developers (both)	15	49,83	22,5	75	15,91	50
Designers (both)	2	40	30	50	14,14	40
Artists (both)	3	42,5	35	50	7,5	42,5
Developers (only)	7	43,21	17,5	62,5	15,98	42,5
Designers (only)	0	N/A	N/A	N/A	N/A	N/A
Artists (only)	3	61,67	47,5	75	13,77	62,5

Table 5.1: Fusion SUS-questionnaire results.

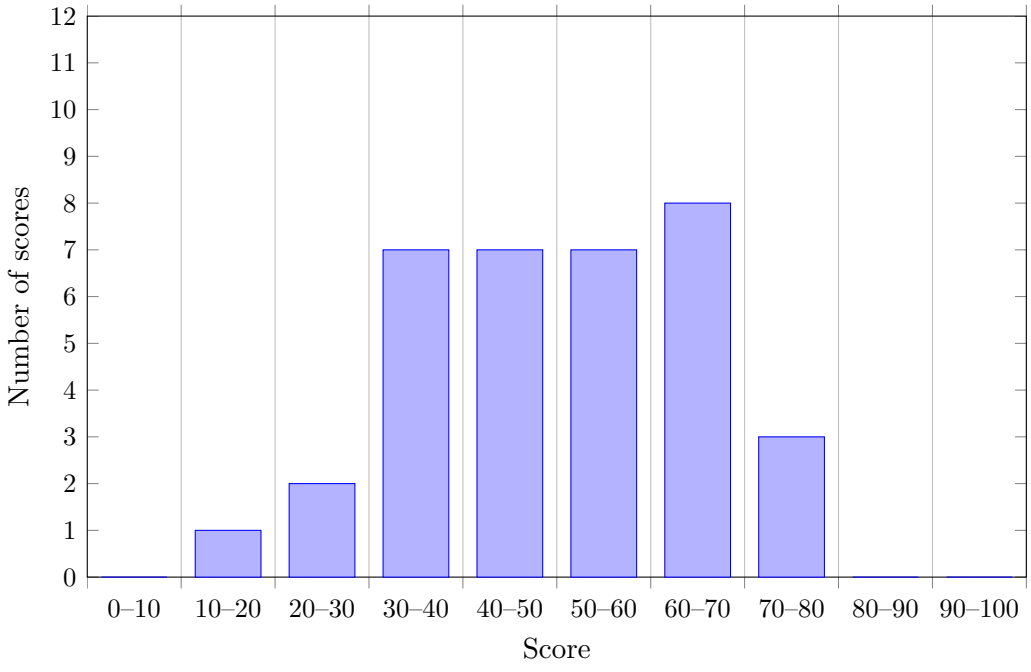

Figure 5.3: Fusion SUS-questionnaire result distribution.

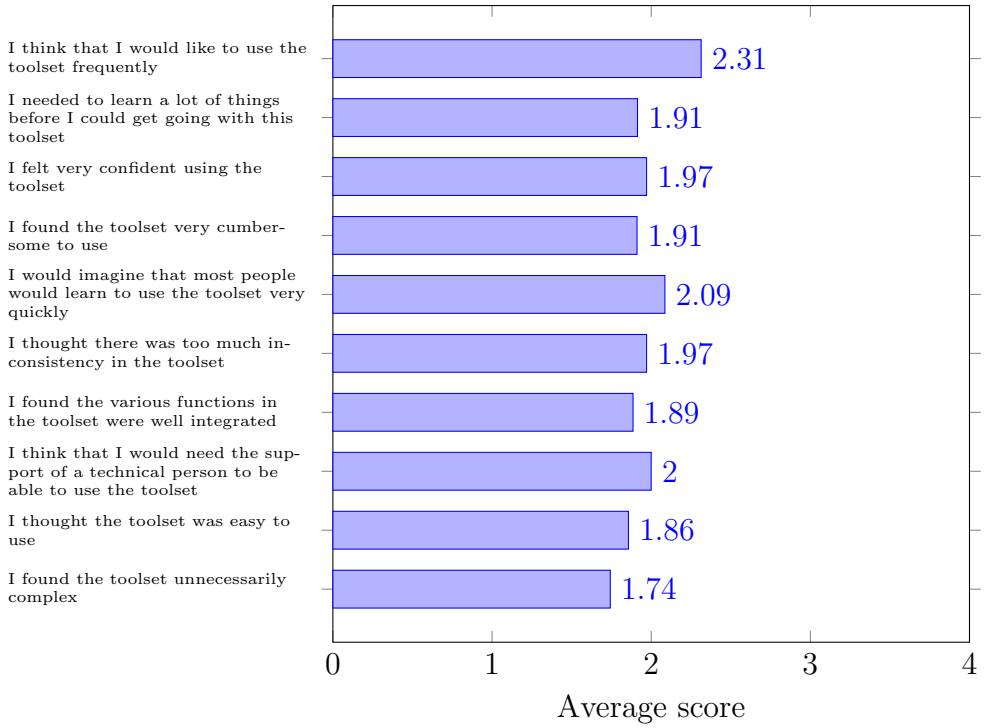

Figure 5.4: Fusion Average SUS-scores by question.

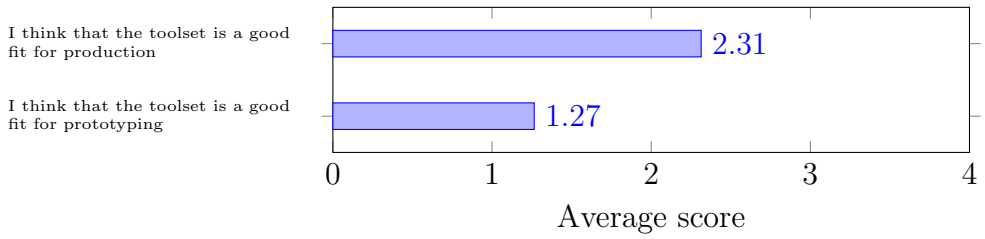

Figure 5.5: Perceived suitability of Fusion for development phases

Feature	N
Unfinished editor	10
Current engine features are unfinished	8
Lack of UI-framework for games	4
3D support missing	4
Lacking animation tools	3
Fragmented environment	2
Poor documentation	2
Slow iteration time	1
Uses C++ and Lua	1
Complex build tool	1

Table 5.2: Criticized features of Fusion.

Feature	N
Uses C++ and Lua	6
Entity-Component system	6
Platform support and build tool	5
Editor	2
Control and transparency of in-house engine	2

Table 5.3: Best features of Fusion.

5.2.2 Unity

Table 5.4 shows that Unity had been previously used by 43 respondents in total. The groups had similar average scores with a few exceptions. Operations scored Unity significantly lower than other groups while designers with both Unity and Fusion experience scored Unity considerably higher. Developers with only Unity experience also scored Unity slightly higher than most of the groups. What is surprising is that compared to the amount of positive feedback Unity received in free-form questions the average SUS-score was relatively low.

The most criticized feature of Unity was MonoDevelop. It was considered both buggy and lacking in features, although it was mentioned that most of the problems can be avoided by using Visual Studio instead. Another common criticism was the difficulty of debugging the scripts, especially on mobile devices, as the debugging never works as expected and it is difficult to set up. Operations thought that the multiplatform-support in Unity is severely lacking, especially when building games for minority platforms. Few respondents also thought that even though there are some quirks in the editor, they are more of a nuisance than a problem. The lack of an UI-framework was also commonly criticized as Unity does not offer that out-of-the-box. There are third party solutions, however, and Unity Technologies is also working on the feature to provide it on a later version. The list of criticized features can be found in Table 5.5.

The most positive feedback was received from designers. The editor received a lot of praise for being easy to use, fast to use for testing ideas and to extend. In addition, it was considered a big plus that designers can do everything inside a single tool. The editor was considered, by far, the best feature of Unity as can be seen in Table 5.6.

Although MonoDevelop was criticized by developers, they considered C# a good and productive language choice for game development, however, some developers wished that Unity would offer support for C++ or access to the source code of the actual engine instead of just to the component scripting interface.

Figure 5.6 shows the distribution of SUS-score for Unity. The distribution of scores shows that even though there is a large group of people scoring Unity above the average score there are also large distribution of scores below the average score. This can also be seen from the relatively large standard deviation in Table 5.4.

The average scores of each individual question in Unity SUS-questionnaire in Figure 5.7 are about 0.5 points higher than in Fusion-questionnaire in Figure 5.4 with a few exceptions. The largest difference is that respondents

found the functionality in Unity considerably better integrated than in Fusion. The other big differences are in the confidence and ease of use of the tool. Again, this was expected as the editor for Fusion, which was predicted to significantly affect these scores, is early in development.

It is somewhat surprising that Figure 5.8 suggests that the respondents found Unity significantly less suitable for production than for prototyping. The score for suitability of Unity for production is only slightly higher than that of Fusion. The suitability of prototyping was expected to be high due to the existence of Unity Editor.

Group	N	Mean	Min	Max	Std. Dev	Median
All	43	63,67	5	97,5	19,09	67,5
Developers	24	67,5	25	97,5	18,24	71,25
Designers	10	63,5	37,5	92,5	17,53	63,75
Artists	4	63,75	45	80	14,51	65
Operations	2	36,25	5	67,5	44,19	36,25
Developers (both)	15	65	25	97,5	22,4	70
Designers (both)	2	83,75	75	92,5	12,37	83,75
Artists (both)	3	64,17	45	80	17,74	67,5
Developers (only)	9	71,67	62,5	85	6,85	72,5
Designers (only)	7	61,43	40	77,5	13,45	57,5
Artists (only)	1	62,5	62,5	62,5	N/A	62,5

Table 5.4: Unity SUS-questionnaire results.

Feature	N
MonoDevelop	7
Building UIs for games is problematic	6
Debugging	5
No native coding	3
No access to source code	2
Editor	1
Licensing issues	1
Working with minority platforms	1
Unclear API	1
Unfriendly data format for VCS	1

Table 5.5: Criticized features of Unity.

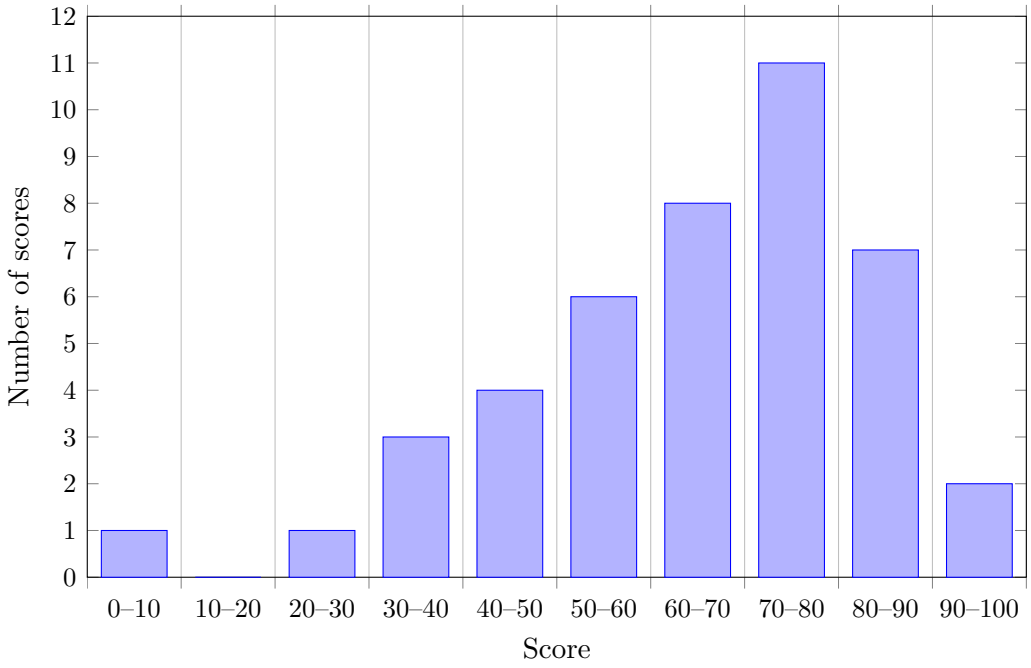

Figure 5.6: Unity SUS-questionnaire result distribution.

Feature	N
Editor	19
C# Scripting	5
Asset pipeline	4
Quick iteration time	4
Third party support	2
Platform support	1
Unified environment	1
Profiling	1

Table 5.6: Best features of Unity.

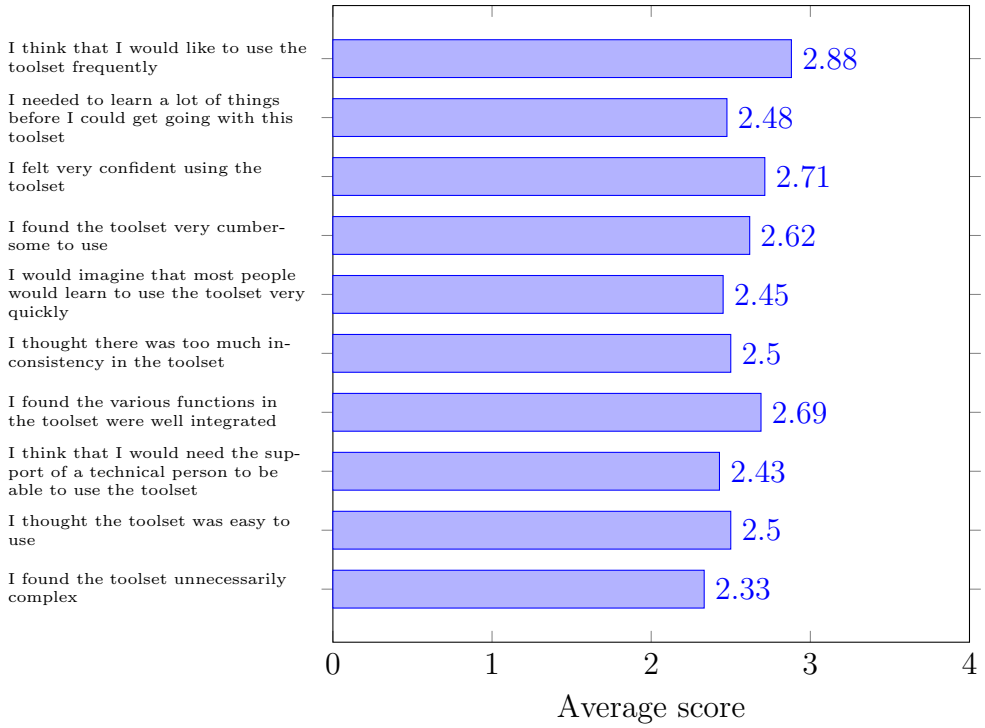

Figure 5.7: Unity Average SUS-scores by question.

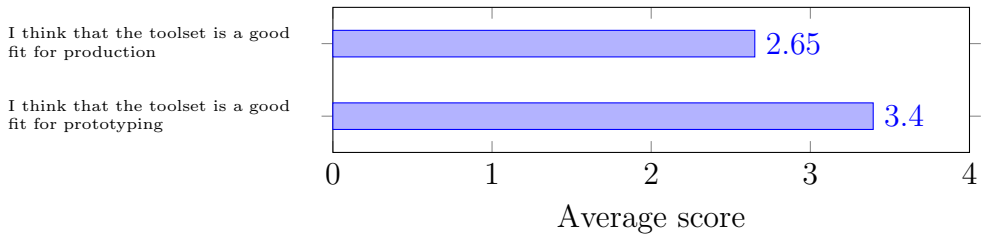

Figure 5.8: Perceived suitability of Unity for development phases

5.3 User Experience for different roles in the team

The members of the chosen game team were interviewed for about half an hour each about the questions in Appendix A. The received feedback supported the results received from the SUS-questionnaire and in addition provided more details on why the scores were so low and how the user experience could be improved.

As cited by four out of seven interviewees the biggest problem in the game development has been the code-driven development culture (see Section 3.6), which caused significant bottlenecks during development of games as most of the changes had to go through the programmers and could not be added directly by other members of the team. A single iteration of adding content or modifying it may have taken hours or even days. More importantly, the content creators say that they did not feel creative ownership over the content they have created as they handed everything over to the programmers who then added it to the game. This culture is slowly changing, but its effects can still be seen especially in Fusion-projects. One of the programmers commented the problems in previous projects, stating that *"in a previous project [the biggest problem] was a huge amount of legacy code on top of which [the features] had to be built."*. According to the designer *"the mindset within [the Case Company] has very much been that when it comes to the pipeline of development the programmers basically control every implementation, which has also resulted in a very one sided team layout where we have far more programmers than people working on creating content or scripts. Everything rests on [the programmers] as well."*

The other problem in the Case Company has been the communication in the Games Unit as cited by three interviewees. According to the members of the game team there were several problems with the communication between the game team and the management of the Unit and the company. The communication with other units was also deemed more difficult than it should be. The level designer described the problem as follows: *"If something is needed it may be difficult to know where to get it or who to talk to about it to get the problem fixed"*. One of the artist had a similar description: *"It is difficult to communicate what kind of tools are wanted and therefore [tool developers] are unable to produce the desired tools. In general the developers and artists speak a different language, and also producers have a different language and so on [sic]."*.

During the interviews the answers to question 5 suggest that in game development there are roughly four different contexts based on the role of the

team member, in each of which user experience of the environment can be evaluated. Programmers concentrate on performing traditional software engineering tasks, i.e. programming and debugging. Artists and level designers create content, e.g. create levels and draw and animate characters. Game designers create rules for the game world and plan the game economy. And all the team members communicate ideas and combine the work together as well as test the game. For example, artist draws and animates a character, programmers write logic that makes the character to jump and to run a specific animation, and finally designer decides on how high the character should jump. All of the previous steps may be done inside multiple tools or in a single tool, but the workflow for each role is different and thus the context of the user experience as well. As expected the interviewees preferred minimal number of different tools, but at the same time they recognized the need for a specialized tool for some of the tasks (e.g. there is no point duplicating the functionality of Photoshop into a game editor). However, when working with multiple tools, integration should be done well so that the workflow would contain as few steps as possible.

Both of the programmers considered efficiency and the fast iteration as the most critical aspect for them. This was expected as technology team, comprised of programmers, had the same opinion (see Section 5.1). In addition general ease of use and stability were mentioned as important aspects. Quoting one of the programmers: *"If I press a button or try to do something then everything has to happen extremely smoothly or I get frustrated."*. The other programmer stated that *"The most important thing is that new assets or logic can be loaded without restarting the game."*

However, both of the programmers in the game team deemed the documentation of the tools and APIs important and having fun while using the environment unimportant in contrary to the opinion of the technology team. According to them, the fun comes from accomplishing their tasks without problems.

Both of the programmers thought that currently the development of Fusion is heading towards the right direction and several good features are implemented, especially in the ECS-framework. One programmer stated that *"the ECS and hierarchy work quite well in Fusion and the bundle-feature is excellent."*. On the other hand they stated that there are still even more features missing or only partially implemented. One of the programmers described the current state of the technology as follows: *"on the technology side [one of the problems] is an unfinished engine with a lot of unfinished and missing features"*, while the designer stated that *"I think we have good technology and good people working on it, but there is much to catch up on what already exists in third party software out there that both the tech team and the*

game development working together on project are always strained on time. Basically one working overtime to catch up, the other working overtime to try to patch the gaps left there. It is something that we have to deal with for a lot longer if we really want to build proprietary tech. The artist working on animations said that *"Now [Fusion] is doing the normal stuff, the basics, stuff moves and stuff scales but it doesn't skew, it doesn't use masks and it doesn't use guides..."*

What the other programmer found most problematic though, was the integration with the systems provided by the Digital Services-team (DS). According to him if there are problems with the SDK provided by DS it may take a long time to get them fixed as they are a separate unit and the communication is difficult. They suggested that the integration work should be made easier through the game engine itself. He described the problems in a following way: *"What slows me down currently the most of all are the problems relating to integration of the [services provided by Digital Services-Unit]. A lot of problems are related to that, for example, lack of communication. It is very hard to get support from their team. And their documentation is missing or at the very least it is out-date if it can be even found and so on."*

One unanticipated finding was that both of the artists and the designer regarded empowerment as the most important aspect. The empowerment did not come up during the workshop with the technology team nor during interviews with the programmers. It seems possible that this is due to the development culture in the Case Company. The programmers already feel empowered but the rest of the team does not. According to the content creators, they have the best user experience when they feel that the environment enables them to create things they could not do before. They also stated that the empowerment improved their feeling of the creative ownership of the game. They said that if they just provide the programmers with the assets the game doesn't feel as much as their own as when they are able to add the assets themselves. The designer described the day when they started to work on the game using the editor: *"When we started to build in the backgrounds and the animations in the backgrounds [the artists] were visibly excited to just see [the ability to work without developers] in effect. It makes you feel like you are a much more relevant part of the development cycle, which is a pretty big thing actually."*

The all three content creators also regarded fun as an important aspect and according to them it comes from the empowerment when they are able to create things and see the results instantly themselves. What was also surprising was that they felt that the documentation is not important. They said that they would rather ask help from people. One of the artists said

that *"[making games can] be fun if there was a lot less export/import reading time...and scrolling through files. I must actually point that out that there is a lot of scrolling and finding."* The another artist said that *"[using tools] is fun if I see results immediately...having fun while working can be seen in the resulting product, especially in artistic things such as games."*.

The designer thought that consistency and unified environment is important. The team members should be able to just commit their changes to version control and demonstrate the work on someone else's computer with the exact same development environment. He said that *"Going from programmers to artists to designers while working with Fusion with very different contexts and capabilities make it very hard to help each other across disciplines if you do not know what the other person's tool is like or behaves like. And that is a scary thing sometimes."*

The producer thought that the usage of the internal technology should be encouraged as the game teams have greater influence over the development of those tools and thus a better experience can be achieved. However, he also stated that with the current resources in the technology team of the Case Company the technology should only be used in few select projects so that the development focus is making a few excellent features instead of several poor ones. He said that *"The people should feel that [Fusion] is built just for them"* and later continued that *"[Fusion] should be worked on together with the game teams so all the developers would feel like 'it was nice to work with Fusion, it was constantly improved during the [game] development process' instead of feeling like 'blah, we must use Fusion [because someone told us to]'"*.

Overall the aesthetics of the tools, documentation and existing knowledge were usually considered the least important aspects and having only a minor impact compared to the other aspects. In addition, the general opinion was that having fun while making the game produces games that are more fun. To quote the producer *"Making fun games should be fun"*.

Interestingly, it was discovered during the interviews that in some game teams the usage of Fusion was a management decision instead of the decision of the game team itself. This was said to have caused friction and to have diminished the experience of using it.

5.4 Observed usage of the tools

The observations in the teams were made by sitting with the members of the game team and observing how they work. They were asked both to mention any problems they encounter as well as to tell if there was some feature that

they felt worked great.

5.4.1 Team using new version of Fusion

The first game team to be shadowed was the one using the new version of Fusion. The team was the first group to start using Fusion Editor outside the technology team and also the first team to have a game in production using the ECS in Fusion. During the observation some members of the team were using Fusion Editor for the first time. Due to the schedule of the game team the observation was done over several days instead of a single day.

First impression of one of the team members installing the editor for the first time was that it feels far more professional than any previous internal tool. He said it was due to the editor having a proper installer, which installs all the necessary dependencies, while the previous tools were stored in SVN, meaning that the tools had to be retrieved from the version control system and all the dependencies manually installed. However, the team used an extension for Fusion Editor which provided specific functionality for creating levels for their game. The installation of the extension required a manual copy of a DLL-file and editing of an XML-file. These manual steps were considered a letdown regarding the positive first impression.

Overall, the team members using Fusion Editor the first time had positive experience when the editor exceeded their expectations. Before using the editor they had had to use text editors to create layouts and scenes in the game instead of doing them visually in the editor. The previous tools that were used internally were also considered poor, so they were surprised how well the new editor worked, taking into account the relatively short time it had been in development.

Shadowing the artists and the designers mostly illustrated the same problems that were already mentioned in the interviews and the SUS-questionnaire. Besides those there were several small specific usability problems such as too complex UI-controls for editing properties, keyboard shortcuts and some drag and drop operations that were not implemented yet, and some operations that required too many steps to perform.

The observation of the programmers was also in line with the results of the interviews and SUS-questionnaire. The only additional problem that was noticed was the increased compile time of the game. The programmers said that in some cases they may need to wait for several minutes for the game to compile. This had slowed the iteration time considerably, although the script components, which do not require compiling and were introduced in the new version of Fusion, had increased the iteration speed earlier. Nevertheless, there are still several things that have to implemented on the native side of

the game, so the increased compile time is a problem for the time being.

5.4.2 Team using old version of Fusion

The second team was using the old version of Fusion. Their game is a sequel to a game series and they are reusing code and tools from the previous game. The game is part of the series discussed in Section 3.6. The focus of the shadowing in this team was on creation of levels and animations, as the problems with the codebase were already well known.

Contrary to expectations, the creation of the levels was very fast. The level designer was able to create a level in a matter of minutes and it could be tested instantly. When questioned, the level designer told that designing levels with the tool is fast because he has done it for a few years using different variations of the same editor in different games. However, the whole editor is based on keyboard shortcuts and the user has to remember all of them as there are no tooltips or working menus. The designer told that for each variation of the editor he has a cheat sheet in his notebook that he uses until he learns the shortcuts.

The level designer was also questioned about how he could add a new type of item to be added to a level or change the properties of existing items. According to him, it is done by manually editing text files and copying files.

In addition, he said that there are several known cases where the editor crashed besides the random cases. He mentioned cases such as zooming too far from the level or trying to test a level without a player character. The crashes were considered minor annoyances as they can be avoided if they are known.

The workflow for adding animations to the game was considered cumbersome. A simple translation animation (i.e. changing a position of an object) where an airship appears from the left side of the screen and flies to the right corner required several steps.

First, an artist draws the object in Photoshop and exports it using a custom script into a custom tool that manages the sprite sheets. In that tool the sprite sheet is created, after which, the sheet is exported into the animation tool. The creation of animation inside the animation tool required, at least in the observed case, a screenshot from the game to get the positioning correct. After the animation is done and saved, a custom program that polls the file system for file changes converts that animation into a format which the game understands. As a final step, to show the animation in the actual game, a developer needs to write a piece of code that runs the animation at a desired moment.

The complex workflow has caused repercussions in the game. For example, the animations sometimes have looked different from what they looked like in the animation tool and the artists have had to double check all the animations in-game. On top of all that the current technical implementation of animations was considered performance intensive and the game team had to limit the number of the animations they have in the game.

5.4.3 Team using Unity

The team using Unity is still on prototyping phase but the prototype is relatively advanced, so the work done should not differ considerably from the work that would be done in production phase. The most significant difference is that the team consists mostly of programmers. Besides programmers there is one person from QA and two artists, one of which acts as a designer as well, while the other is a technical artist. Overall, the members of the team are more experienced than members of an average game team. Due to the technically-oriented and experienced nature of the team the observations were focused on tasks related to programming.

As with the other teams, the shadowing showed mostly the same problems that were already mentioned in the feedback of the SUS-questionnaire. One of the most interesting observations was related to a problem that Unity has with the assets and version control systems. According to the team it is practically impossible to work on a scene simultaneously as the serialization format of Unity does not work well with VCSs. The team is using something they call the "hat system". If someone is modifying a scene he has to take the hat and if someone else needs to make changes as well he has to find the person with the hat and work together with him.

When people with different roles worked together they sat down together on one computer. For example, when an artist and a programmer worked on creating a new effect for the game the artist first described the desired effect to the programmer and told him where to find the art necessary for it. Then they sat down and the programmer started to iterate on the effect while the artist gave feedback on each iteration. For the most part this way of working seemed to work well, but every time the art needed some changes the artist had to move back to his own computer to make them and then commit them to the version control before the programmer could get the new version to his computer.

Overall, it seemed that the most of the functions in the Unity Editor work well. There were several smaller problems that the team found extremely annoying. For example, saving changes to a *prefab* (i.e. a game object with pre-defined components and properties) requires saving a scene

as well. The team said that they had lost changes several times as after clicking the "apply"-button in the prefab to save the changes to it, they also have to remember to save the scene. They also felt the whole operation is unreliable and thus when saving a prefab they clicked apply several times while simultaneously pressing Ctrl+S on the keyboard to save the scene multiple times as well. It was also unclear what the component properties of prefabs shown in bold meant. Supposedly, they were meant to show the values that have changed since the last save, but even after saving both the scene and the the prefab, some of the properties in the editor were still shown as bold.

The team used logging for debugging the game. As reported already in the SUS-questionnaire, the debugging does not work well within Unity. During the observation the programmers debugged the scripts by adding logging lines all over the scripts to debug the values in the different parts of the script. As an additional minor annoyance, there was no log rotation (i.e. everything is logged into a single log-file) which caused that the opening of the log-file took a long time and even crashed some text editors.

Here are some other examples of problems mentioned by the team: Unity using old version of Mono-compiler, shaders working inconsistently between platforms and having to create a workaround to pause the editor during mouse input. In general, the team expected that a product that has been in development for over ten years should be a lot more polished.

5.5 Fusion design and implementation

The results of this study will be utilized in the design and implementation of Fusion in the Case Company. Both the editor and the engine have been in development before this study began so the results both validate (or invalidate) the current implementation and aid in design, implementation and prioritization of the future features.

Improvements are planned for most of the problems uncovered during this study, but only a few are currently implemented. For example, a batch-file was created for the installation of an editor extension. This was a fast and easy way to remove the manual steps from the installation process and it improved the process considerably. A second example was adding a shortcut for navigating between different files in the game project. By pressing Ctrl+G the users get a pop-up control that allows them to directly type the name of the file they want to navigate into by using search strings. This small change removed the need for navigating the files and folders using a traditional tree-hierarchy for the most part, with the exception of working with an unfamiliar project when the user may not necessarily know the names of the files.

Another example of an implemented improvement came during the shadowing of the first team, where one of the artists was creating particle effects with the Fusion Editor. The artist thought that the real-time editing was a big step up from her previous workflow where she had to edit those properties with a text editor and restart the game to see the changes. However, she found the UI-controls for editing different properties of the particle systems still a bit too complex. The problem was discussed with her and the author of the particle system engine implementation. With the additional information on the use cases from the artists and the technical solution from the author the UI-controls were simplified considerably in a few iterations. The iterations of the controls can be seen in Figures 5.9, 5.10 and 5.11. The validation of a successful implementation came a few days later when the artist wanted to explicitly thank the technology team for building the editor and told that she enjoyed using it.

Listing 5.1 shows the data structure of a single particle system which the users had to manually edit in text-files containing one or more of these systems in addition to other data. Also, when users wanted to test the changes they had to save the file and restart the game to see them, although later *hotloading* (i.e. loading changes on the fly) was added to be able to reload the data while running the game. Also as can be seen from Listing 5.1 the properties in the data are not arranged consistently. For example, particleEndColor-property is before particleStartColor-property and there are several unrelated properties between them. The order was not fixed so the users could rearrange the properties manually but it was still a lot of work to find the desired properties from files. In addition, the format of the properties is unclear, as some of the properties have an array of a string and a number, while others have an array of a string and an inner array of numbers. In both cases the first item represents the type of the property while the second represents the value. The users had to know a different list of valid strings for the first item in each property as well as depending on the first item if it is a single number or an array with a specific size. For example, "`LINE_2D`" requires an array of four numbers; first two numbers are the starting point of the line (i.e. x_1 and y_1) and the last two numbers are the end point (i.e. x_2 and y_2). With "`LINE_3D`" there would be an array of six numbers(i.e. x_1, y_1, z_1, x_2, y_2 and z_2 respectively.

Figure 5.9 shows part of the development version of the Particle System editor. This version was only used to test that the technical implementation works and was never given to users. The biggest benefit of the development version was to move the editing into the editor instead of text-files and the properties of the system could be edited during runtime and the changes could be seen immediately instead of having to restart the game.

Figure 5.10 shows the version that was first given to users. This version of the editor shows the list of all types and changes the number of number value boxes depending on the selected type. It did not filter the types depending on the property so the user still had to know which types were valid for each property. In addition, user also had to know what the values represented as it only showed the order of the values instead of the name of the value. In both this version and the development version the properties were ordered alphabetically instead of ordering them according to their behaviour.

The latest version of the particle system editor so far can be seen in Figure 5.11 which contains several improvements over the previous versions. Most of the properties could be simplified into sliders representing either a constant value or a range between specific values. In addition, the properties were grouped and arranged logically and tooltips were added for each property. The properties which could not be simplified into sliders were left mostly similar to the ones in Figure 5.10 with the exception of the types being filtered to show only valid ones and the values having proper names (i.e. x_1 and y_1 instead of [1] and [2]). Also the properties regarding colors were changed to use color picker-controls similar to ones in image editing applications such as Adobe Photoshop. This simplification of the controls was possible as a result of getting more detailed information about the use cases for the particle system directly from the game team.

The changed controls in Figure 5.11 changed the way artists work in the game team. Instead of thinking of which values would produce a desired effect the artists would just move sliders and see how the effect changed in real-time. This change accelerated the development speed as it reduced the mental load by removing the need of remembering what each property did, allowing users to just try out changing the value using the slider instead and see the result. Also, as the properties were arranged logically there was considerably less time spent on searching for a desired property. The team said that the changes lead to artists being able to produce significantly better effects in the game than before.

```
"emissionGravity": [
  "CONSTANT",
  30
],
"emissionLifeTime": [
  "CONSTANT",
  -1
],
"emissionLimitKill": "none",
"emissionMaxParticles": [
  "CONSTANT",
  36
],
"emissionPosition": [
```

```
    "LINE_2D",
    [0,0,0,0]
  ],
  "emissionRate": [
    "CONSTANT",
    12
  ],
  "emissionResetInterval": [
    "CONSTANT",
    0
  ],
  "emissionRotation": [
    "CONSTANT",
    -1
  ],
  "emissionRotationSpeed": [
    "CONSTANT",
    0
  ],
  "emissionStartTime": [
    "CONSTANT",
    0
  ],
  "emissionStopTime": [
    "CONSTANT",
    0.1
  ],
  "emissionVelocity": [
    "CONSTANT",
    0
  ],
  "emissionVelocityFactor": [
    "CONSTANT",
    0.9
  ],
  "emitterAmount": [
    "CONSTANT",
    32
  ],
  "isRadial": false,
  "isWorldSpace": false,
  "particleEndAlpha": [
    "CONSTANT",
    0
  ],
  "particleEndColor": [1,1,1,1],
  "particleEndSize": [
    "CONSTANT",
    0.8
  ],
  "particleGravity": [
    "POINT_2D",
    [50, 400]
  ],
  "particleLifeTime": [
    "RANGE",
    [0.5, 0.8]
  ],
  "particleSpriteEndRotationSpeed": [
    "RANGE",
    [0, 68]
  ],
```

```
"particleSpriteRotation": [
  "RANGE",
  [0, 360]
],
"particleSpriteStartRotationSpeed": [
  "RANGE",
  [100, 318]
],
"particleStartAlpha": [
  "CONSTANT",
  1
],
"particleStartColor": [1,1,1,1],
"particleStartPosition": [
  "BOX_2D",
  [-20, 20, 20, 20]
],
"particleStartSize": [
  "CONSTANT",
  0.8
],
"particleStartVelocity": [
  "BOX_2D",
  [-350, -25, 350, -320]
],
"particleWind": [
  "POINT_2D",
  [0, 0]
],
"shader": "ptc",
"spriteFrameRate": [
  "CONSTANT",
  10
],
"sprites": [
  "PARTICLE_BLOCK_MUD_1",
  "PARTICLE_BLOCK_MUD_2",
  "PARTICLE_BLOCK_MUD_3",
  "PARTICLE_BLOCK_MUD_4"
],
"textureAnimation": "randomStart",
"visible": true
}
```

Listing 5.1: Particle System Data

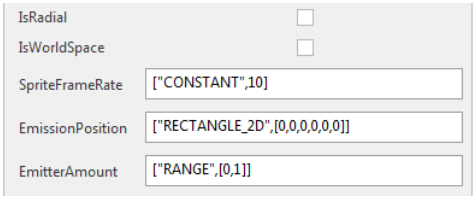

Figure 5.9: The development prototype of Particle Editor.

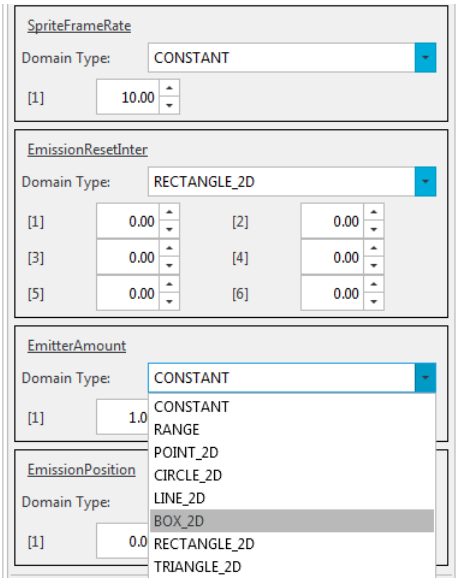

Figure 5.10: The first version of Particle Editor.

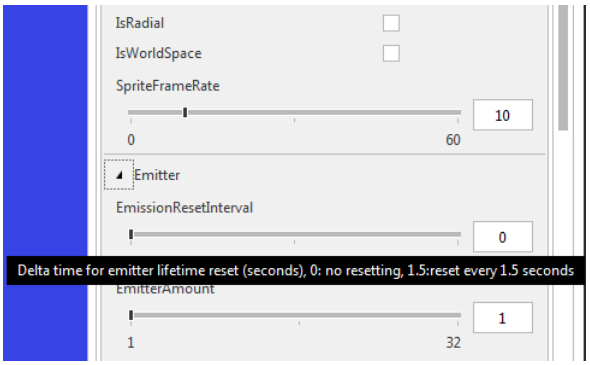

Figure 5.11: The latest version of Particle Editor.

Chapter 6

Evaluation

This chapter describes how well the research objectives were achieved. In addition, the effects of the research in the Case Company are discussed.

6.1 Research Objectives

As the results in Section 5.5 show, the current development tools in the Case Company can be improved significantly. The starting point for working with the particle effects was the manual editing of the text files and restarting the game after each change, while the current version enables real-time editing using sliders and color pickers. This not only accelerated the development but also changed the whole way of working from thinking about the values that would produce a desired effect into trying out different ideas by adjusting the values and seeing which idea looks the best. In other words, earlier the artists thought about the effect that they wanted to produce and tried to come up with the values that produce it, whereas they just start working on the effects without having a clear final goal and see what they come up with while trying out different ideas.

The artists save a considerable amount of time by using the editor instead of text-files to edit the particle effects. Some effects may contain several systems with different properties and thus the resulting data is multiple times larger than the data shown in Listing 5.1. The more complex the effect the more time is saved. It can be argued that by using the text files the creation of complex effectis tremendously more time consuming. With simple additions, such as the quick navigation, a user also saves a few seconds for each navigation operation. With more than a hundred users and several operations per day those seconds quickly add up to hours and days. With these results it can be safely said that the main objective of this research

was achieved, although these are just few of the many things that can be improved in the development tools in the Case Company. This result is also validated by having users coming to thank the editor team for building the editor.

The results in Section 5.2 show that the level of usability of the development tools in the Case Company leaves a lot to be desired. Especially the SUS-score of Unity is surprising compared to its popularity among game development studios. This would suggest that there is no significant correlation between the level of usability and the adoption of specific tools. However, this may also mean that even though the usability of Unity is low it may still beat most of the other tools in terms of usability and for that reason it is popular in mobile game development. It would also mean that there is a lot of room for improvement in game development tools overall.

The results from the interviews in Section 5.3 show that the game teams think that to produce games that are fun require that their development is fun as well. Validating this statement was out of the scope of this thesis as it would require a long-term research into development of a game from concepting to post-production as well as gathering data from the players of the game.

Overall, the objectives set in Section 1.3 were achieved. The main objective was achieved with clear results while the results for the other two objectives can only be considered preliminary.

6.2 Changes in technology development in the Case Company

As the study involved meeting and discussing with the members of the game teams the visibility of the technology development improved in the Case Company. The game teams are now slightly better aware of the work done in the technology team and know that they can affect the development of the technology. In addition, there were some changes that happened in the way the technology team works during the study, some partly due to either the study itself or its results.

The biggest change is that the technology team now engages in more interaction with the game teams. This enables the technology team to know the needs of the other teams better and to lower the threshold for getting feedback straight from the users. Also, one designer decided to take part in the weekly meetings of the Fusion Editor-development team to provide feedback on the implemented features.

The results show that the UX of the internal tools is severely lacking and the team is currently trying to find a solution to improve the situation. Several small improvements are already made to the technology, but the team considers that a permanent solution has to be found for them to be able to consistently provide high quality solutions. The interaction with the game teams contributes to delivering better quality technology than before.

Chapter 7

Conclusion and Discussion

This chapter combines the results of the methods and discusses how the game development environments in the Case Company could be improved.

7.1 Validity and reliability

The target organization of the study was a single Finnish mobile game company, and although the research involved data gathered from several game teams, the way of working inside the same company is similar. This should be taken into account with the results of this study as the game industry is large and it contains a large amount of different organizations, each with their own culture and technologies.

In addition, the concept of both Fusion (the new version) and Unity are similar. Both environments are based on users creating scenes where game objects are placed and where the game objects contain behavior components. An additional study to compare environments with different concepts would be required to see if the basic concept of the environment has any effect on the user experience.

The results of a recent study by Kasurinen et al. suggests that game development organizations are pleased with their current technical solutions (Kasurinen, Strandén, and Smolander 2013). The study was done in South-Eastern Finland and included seven organizations with varying levels of experience (from startups to companies with more than ten published products). With the exception of one organization, all of the organizations were planning to use a third party engine in the future. The results of that study contradict the results in this thesis as the results in the Case Company suggest that the game teams are not content with the current tools.

Another study, conducted Petrillo et al., found that the technological

problems were cited in 60% of the projects, while tool related problems were cited in 35% respectively (Petrillo et al. 2009). The result suggests that there are problems in the technology and the development tools and contradicts the results by Kasurinen et al. There is a four year gap between the studies but it is unlikely that the industry could have managed to solve the problems during that time. Kasurinen et al. also identified this contrast between their results and the results of the previous studies (Kasurinen, Strandén, and Smolander 2013).

It should also be taken into account that the ECS-framework and the editor for Fusion were both in early development and were not considered production-quality (even though there is a pilot project in production that uses them). This has an obvious impact on the results as seen by low scores and large number of reported problems. It can be expected that with an improved version of the Fusion Editor the usability and the UX of Fusion is greatly improved.

The SUS-questionnaire yielded similar results for all the groups of a single environment. A more specific grouping may have been necessary to get more detailed results from the questionnaire. For example, the questionnaire could have included separate SUS-scoring for different parts of the environment (i.e. scoring API, editor, build tool etc. separately) and also more questions about the past experience of the user (i.e. how long the user has used the environment and what other environments he has used besides Unity or Fusion). This additional information may have helped analyze some specific features.

Different phases of game development have different requirements and different focus. The results suggest that some tools are significantly better for different phases than others. The interviews and the shadowing should have had more focus on determining these differences in the development phase to be able to analyze the factors that cause the difference in suitability of the tool.

Informal chats with the members of the game teams during the research revealed that there are a lot of strong subjective opinions about different technologies. For example, some programmers disliked Unity just because they prefer using C++ instead of C# even though objectively measured they could be more productive using Unity. In addition, due to the development culture of the Case Company and the pain some teams have gone through with the old games and the old technology there were people that considered Unity as kind of a "holy grail" that would solve all the past problems. Further research would be required to determine how much these prejudices and biased opinions affected the results of the SUS-questionnaire and also to determine how much they affect the overall user experience.

Even though the members of the game team felt that having fun while

developing games produces more fun games (see Section 5.3), it should be measured if this is the case. The measurement is challenging as fun is a subjective term and it would be difficult to measure the size of the effect of the development environment on a single game. However, it can be hypothesized that enjoying the game development results in more motivated developers, which in turn causes the developers to put more effort into developing the game.

7.2 The current state in the Case Company

According to Bangor et al. a product should have SUS-score above 70 to be considered acceptable and any product with less than 50 are judged unacceptable and should raise significant concern (Bangor, Kortum, and Miller 2008). The results in Section 5.2 show that the usability of Unity is considered marginal while the usability of Fusion is unacceptable.

The rating for Unity is surprising, as it does not correlate with the high adoption of Unity in mobile game companies. The difference in scores between Fusion and Unity is also quite small even though Unity has been in development since 2001 while Fusion for only about three years. It can be hypothesized that the game teams have higher expectations for Unity than for Fusion and that affects the score, or that the strong subjective opinions encountered during the study lower the score for Unity.

In Social & Mobile Technology Survey in 2012, Deloura reported that rapid development time was the most important engine feature (Deloura 2012). The results in Sections 5.1 and 5.3 show that it is also one of the most important aspects of the user experience of the development environment. It can be argued that empowerment, which was reported as the most important aspect for content creators, consists at least partially of the ability to make changes rapidly.

The data-driven component-based system for building games used by both Unity and Fusion is a proven concept in the game industry (Bilas 2002; McShaffry 2013). The problems of past code- and programmer-driven culture of the Case Company are slowly disappearing with the usage of the current development tools. The basic concept of the system, i.e. creating small individual components, prevents the birth of problematic codebases, which are common in the older games of the Case Company. This can be expected to improve the current situation considerably.

In addition, Fusion already implements some improvements over Unity which are positively received by the game teams. For example, Fusion uses JavaScript Object Notation (JSON) to store the data which works well with

the version control systems and is human-readable so it can also be edited by hand compared to the proprietary format used in Unity.

However, the old tools seem to already enable fast iteration in some cases as proven by the level editor mentioned in Section 5.4.2. It suggests that although the tools may have severe usability problems they still meet the most important requirements for the game teams. This indicates that the tools may be provided for the game teams in an early development phase even if they implement only some of the most important features.

So far, the game teams have not been an active part of the tools development in the Case Company. Even though the technology team is using Scrum for the most part as defined by Sutherland and Schwaber (Sutherland and Schwaber 2011), the team is missing a product owner completely. Currently the team creates and prioritizes the stories themselves. This lack of user involvement probably has a negative effect on the user experience as the technology team itself is not using the tools to make games.

7.3 Improving the current state

According to Folmer, the game development tools may require twice the amount of code compared to the game itself (including the engine) and are an important part of the development due to the number of content creators game teams have currently (Folmer 2007). In contrast, the technology team has only three game tools programmers serving all the game teams in the Case Company. It could be argued that a large part of the problems in the internal tools is due to either lack of resources or having a too big scope by serving several game teams with different games at the same time.

The study shows that there is a lot of work to be done on polishing the existing features and getting the usability of those features on an acceptable level. This would obviously mean that either the implementation of missing functionality gets delayed, the technology team has to work with fewer game teams (to reduce the amount of features needed) or the technology team gets more resources in Fusion development. The results of the research strongly suggest that unless the usability of Fusion is improved the game teams will become less willing to use it.

In addition, the UX should be more integrated into the development process. Hartson and Pyla suggest an approach where there are simultaneous UX and SE-sprints where the workflow is alternated (Hartson and Pyla 2012). The basic idea is that the UX-team creates designs and paper prototypes with the customer (i.e. game team in this case) as well as evaluates the implemented features, while the SE-team is implementing the designs by the

UX-team in the previous sprint.

The study suggests that, at least in the case company, the sweet spot for the game development environment is somewhere between Unity and the current state of Fusion. This can be deduced from the facts that the developers prefer the use of internal and native (i.e. C++) technology as it gives them more control and performance, while the designers and artists prefer unified environment of Unity and the focus on the GUI-tools.

The past experience in the industry suggests that the tools should operate within the game engine to enable fast iteration (Schaefer 2000). The game team described in Section 5.4.2 had similar problems as Schaefer, although on a smaller scale, when they could not work with the animations directly in the game.

One of the observed game teams (see Section 5.4.1) had problems with increased build times. According to Blow slow build times may create several minute long delays before the changes are seen in the new version of the game (Blow 2004). This can be solved by either finding a solution to speed up the build times or by scripting. Currently the technology team is working on both with more focus on scripting. The scripting is a proven method to enable very fast iteration speed, and Bilas, for example, states that being able to run scripts while running the game enabled them to add content to the game very quickly (Bilas 2000).

During the observation of the team using Unity (see Section 5.4.3) one of the team members wondered why the hotkey for scale is 'r' while the hotkey for rotate is 'e'. According to him it would make more sense that the 'r' is rotate. However, according to Acton the tools should imitate the functionality in Autodesk Maya (Acton 2013). The hotkeys in question in Unity are the same as in Maya. As Maya is commonly used in the game development it makes sense to have the same hotkeys, but in this case, as the user has no previous experience in Maya, the hotkeys did not make sense. This can potentially be remedied by giving the user a possibility to change the hotkeys, although the defaults should still be the same as the other tools, because it reduces the amount of time to learn the different hotkeys between different environments.

The technology team arranges an event called Tech Summit which occurs about every to to three months and lasts for about a day or two. So far, this has been the only arranged event for the game teams to see and hear what is happening in the technology development. The event consists of presentations about selected topics. Currently, there haven't been any workshops arranged. In comparison, Ahern states that Insomniac Games arranges an Usability Week which allows people from different teams and roles to use the tools and get actionable feedback from them. According to him, they

originally held it only for a day, but found out that it wasn't nearly enough to get actionable information. In addition, he also noted that they have an open house event where the users can meet the developers and try out the tools in development. Ahern says that it has helped greatly to close the communication gap between the departments (Ahern 2012).

The results in Section 5.5 prove that by interacting with the game teams the user experience can be improved considerably. Obviously, elicitating requirements from the users is part of normal requirements engineering process (Nuseibeh and Easterbrook 2000). However, it can be argued that a close personal engagement with the users improves the experience even further, indicating that the experience could be improved by developing the tools as part of the game team instead of in a separate group. This way the tools could be made even more specific to the game. Some organizations have a separate team that creates a framework for the tools while the game teams have either their own game tools programmers or the game programmers build the tools (Vachon 2013).

In his GDC-presentation, Lightbown highlights that the development tools can save a lot of money by reducing just a small amount of time required by a developer to perform some common task (Lightbown 2013). By saving 20 minutes a day per developer in an organization with 200 developers the organization saves 100 months a year. The amount of money saved in Finland is about 500 000 € (using an estimate cost of 5000 € per developer, the average salary in Finland is 3590 € according to Tekes (Tekes 2012)). The quick navigation feature mentioned in Section 5.5 is a good example of a feature that saves a small amount of time that quickly adds up.

7.4 Future work

The latest version of Unreal Engine was recently voted as the one of the most important technologies used in games[1]. The development kit for Unreal Engine, UDK 4, offers not only the editor but also source code for the whole engine. Unity received most criticism from developers for not having C++-support or source code access for the engine, while Fusion was criticized by content creators for not having a proper editor. It would be interesting to study how UDK 4 is perceived by the users of different roles as it does not share the shortcomings of either of the two different environments used in this study.

[1]http://www.develop-online.net/news/the-top-100-tech-in-games-revealed/0194694, retrieved on 2.7.2014

The study by Kasurinen et al. identified a need for a tool being able to add notes to the items in the levels of the game to inform the other members of the team about the exact changes required for that item (Kasurinen, Strandén, and Smolander 2013). Similar observations were also made during this study, as the team using Unity had to use their own "hat system" to communicate within the team. It could also be argued that the need for a unified environment identified during the interviews is due to need for being able to better communicate within the team.

Hölttä states that the need for increasing the role of conscription devices in engineering communication is evident (Hölttä 2013). The items in the game such as levels, characters and objects can be considered as conscription devices and it would be beneficial for the team to be able to facilitate the communication by adding notes to those items to inform the other members of the team what additional actions are necessary for those items. One possible future direction for this study would be to investigate how the tools could improve the workflow in the game teams by integrating at least some parts of the communication into the development tools.

Another future direction for the research is to study how the tools could enable testing the user experience of the game early in development. It has been identified that game studios start testing for UX of games too late in the development life cycle, sometimes as late as beta, which means that most of the feedback obtained from the tests is unlikely to have an impact on the final game (McAllister and White 2010).

References

Acton, Mike (2013). *Usability Is Not Random*. URL: http://www.gdcvault.com/play/1017953/Usability-Is-Not.

Ahern, Sean (2012). *It Stinks and I Don't Like It: Making a Better Engine Experience At Insomniac Games*. URL: http://www.gdcvault.com/play/1015729/It-Stinks-and-I-Don.

Bangor, Aaron, Philip T Kortum, and James T Miller (2008). "An empirical evaluation of the system usability scale". In: *Intl. Journal of Human–Computer Interaction* 24.6, pp. 574–594.

Bevan, Mike (2013). *The SCUMM Diary: Stories behind one of the greatest game engines ever made*. URL: http://www.gamasutra.com/view/feature/196009/the_scumm_diary_stories_behind_.php.

Beyer, Hugh and Karen Holtzblatt (1999). "Contextual design". In: *interactions* 6.1, pp. 32–42.

Bilas, Scott (2000). *Postmortem: Sierra Studios' Gabriel Knight 3*. URL: http://www.gamasutra.com/view/feature/3127/postmortem_sierra_studios_.php.

— (2002). "A data-driven game object system". In: *Game Developers Conference Proceedings*. URL: http://scottbilas.com/files/2002/gdc_san_jose/game_objects_slides.ppt.

Blow, Jonathan (2004). "Game development: Harder than you think". In: *Queue* 1.10, p. 28.

Brooke, John (1996). "SUS-A quick and dirty usability scale". In: *Usability evaluation in industry* 189, p. 194.

Caudle, Sharon L (2004). "Qualitative data analysis". In: *Handbook of practical program evaluation* 2, pp. 417–438.

Deloura, Mark (2012). "Social & Mobile Technology Survey 2012". In: *Game developer magazine*.

Folmer, Eelke (2007). "Component Based Game Development–A Solution to Escalating Costs and Expanding Deadlines?" In: *Component-Based Software Engineering*. Springer, pp. 66–73.

Gamasutra (2014). URL: http://www.gamasutra.com.

GDC Vault (2014). URL: http://www.gdcvault.com.

Gregory, Jason (2009). *Game Engine Architecture*. A K Peters/CRC Press.

Hartson, Rex and Pardha S Pyla (2012). *The UX Book: Process and guidelines for ensuring a quality user experience*. Elsevier.

Hölttä, Venlakaisa (2013). "Beyond Boundary Objects – Improving Engineering Communication with Conscription Devices". PhD dissertation. Aalto University.

Hyysalo, Sampsa (2009). "Käyttäjä tuotekehityksessä: tieto, tutkimus, menetelmät". In: *Taideteollisen korkeakoulun julkaisu B* 97.

ISO (2010). "9241-210: 2010. Ergonomics of human system interaction-Part 210: Human-centred design for interactive systems". In: *International Organization for Standardization (ISO), Switzerland*.

Kasurinen, Jussi, Jukka-Pekka Strandén, and Kari Smolander (2013). "What do game developers expect from development and design tools?" In: *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. ACM, pp. 36–41.

Law, Effie Lai-Chong et al. (2009). "Understanding, scoping and defining user experience: a survey approach". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, pp. 719–728.

Lightbown, David (2013). *The User Experience of Game Development Tools*. URL: http://gdcvault.com/play/1019273/The-User-Experience-of-Game.

McAllister, Graham and Gareth R White (2010). "Video game development and user experience". In: *Evaluating User Experience in Games*. Springer, pp. 107–128.

McShaffry, Mike (2013). *Game coding complete*. Cengage Learning.

Nielsen, Jakob (2012). "Why you only need to test with 5 users, 2000". In: *Jakob Nielsen's Alertbox. Available on line from www. useit. com/alertbox/20000319. html*.

Novak, Jeannie (2011). *Game development essentials: an introduction*. Cengage Learning.

Nuseibeh, Bashar and Steve Easterbrook (2000). "Requirements engineering: a roadmap". In: *Proceedings of the Conference on the Future of Software Engineering*. ACM, pp. 35–46.

Petrillo, Fábio et al. (2009). "What went wrong? A survey of problems in game development". In: *Computers in Entertainment (CIE)* 7.1, p. 13.

Schaefer, Erich (2000). *Postmortem: Blizzard Entertainment's Diablo II*. URL: http://www.gamasutra.com/view/feature/3124/postmortem_blizzards_diablo_ii.php.

Sididris, Romain (2014). *AAA Mobile Games Development: Challenges*. URL: http://www.gdcvault.com/play/1020486/AAA-Mobile-Games-Development.

Sutherland, Jeff and Ken Schwaber (2011). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*.

Tekes (2012). *The Finnish game industry*. URL: `http://www.slideshare.net/TekesICT/the-finnish-game-industry`.

Vachon, Pierre-Luc (2013). *Assassin's Creed III Homestead: Code, Data, and Tools*. URL: `http://www.gdcvault.com/play/1017933/Assassin-s-Creed-III-Homestead`.

Appendix A

Interview Questions

Question 1: Current role, responsibilities, past experience etc.

Question 2: In your opinion what is the biggest problem or the bottleneck in the game development? (i.e. bad tools, unrealistic schedules or scopes, feature creep, bad communication...).

Question 3: Which tools do you now you use?

Question 4: Do you think that better tools, compared to the current ones, could improve your daily work significantly?

Question 5: Describe your ideal Game Development Environment. *Describe the whole workflow of performing one of the most common tasks you do in your daily work* (i.e. adding new animation, new feature etc.)

Question 6: What are the best features of the current tools? Could those be improved further?

Question 7: Which important features are the most lacking or completely missing from the current tools? Why are they important?

Question 8: Have you used Unity? (if answer is positive, repeat questions 6 and 7 for Unity toolset)

Question 9: What are the most important features of a game development environment for you? Why?

Question 10: What is the best tool you have used? (Can be as simple as a text editor or something more complex) How it relates to game development environment?

Question 11: How would you improve the current tools?

Question 12: Can the usage of the tools be fun? What things make it fun?

Question 13: If you're having fun making the game will it have a great impact in the end-product.

Question 14: Would you like to be more involved in the development of the tools?

Question 15: Order the following aspects from the most important to the least important.

- Fast iteration time
- Ease of use / Simplicity
- Stability
- Look and Feel
- Fun to use (attracts you to create and try out several things)
- Available knowledge (i.e. community, forums etc.)
- Documentation
- Frequent updates
- Ability to affect the development of the tool

Appendix B

Games Unit Questionnare

Question 1: What is your role in the game team?

Question 2: Which applications do you use in your daily work?

- Visual Studio
- MonoDevelop
- Text Editor (i.e. Sublime Text, Notepad++, Vim etc.)
- Photoshop
- Flash
- Fusion Editor
- Unity Editor
- Other

Question 3: Which of the following parts of Fusion toolset have you used?

- Fusion Engine C++ API
- Fusion Engine Lua API
- Fusion Engine ECS API
- Fusion Editor
- Build tool
- Photoshop Exporter
- Flash Exporter
- Art Packer

Question 4: Which of the following parts of Unity toolset have you used?

- Unity Scripting

- Unity Editor
- Build tool
- 3rd Party Plugins

Question 5&9: Rate the following statements about [Fusion,Unity] toolset (i.e. the engine API, editor, build tools, art pipeline etc.) from one to five.

- I think that I would like to use the toolset frequently.
- I found the toolset unnecessarily complex.
- I thought the toolset was easy to use.
- I think that I would need the support of a technical person to be able to use the toolset.
- I found the various functions in the toolset were well integrated.
- I thought there was too much inconsistency in the toolset
- I would imagine that most people would learn to use the toolset very quickly.
- I found the toolset very cumbersome to use.
- I felt very confident using the toolset.
- I needed to learn a lot of things before I could get going with this toolset
- I think that the toolset is a good fit for prototyping
- I think that the toolset is a good fit for production

Question 6&10: What are the best features of [Fusion, Unity] toolset. Why? Please specify up to three features.

Question 7&11: Which features should be improved in [Fusion, Unity] toolset. How? Please specify up to three features and a short description how those features could be improved.

Question 8&12: Which features are completely missing from [Fusion, Unity] toolset. Why are they important? Please specify up to three features.

Question 13: Any additional feedback?